Learning Distributed and Fair Policies for Network Load Balancing as Markov Potential Game

Zhiyuan Yao*
École Polytechnique, Cisco Systems
zhiyuan.yao@polytechnique.edu

Zihan Ding*
Princeton University
zihand@princeton.edu

Abstract

This paper investigates the network load balancing problem in data centers (DCs) where multiple load balancers (LBs) are deployed, using the multi-agent reinforcement learning (MARL) framework. The challenges of this problem consist of the heterogeneous processing architecture and dynamic environments, as well as limited and partial observability of each LB agent in distributed networking systems, which can largely degrade the performance of in-production load balancing algorithms in real-world setups. Centralised-training-decentralised-execution (CTDE) RL scheme has been proposed to improve MARL performance, yet it incurs – especially in distributed networking systems, which prefer distributed and plug-and-play design schemes – additional communication and management overhead among agents. We formulate the multi-agent load balancing problem as a Markov potential game, with a carefully and properly designed workload distribution fairness as the potential function. A fully distributed MARL algorithm is proposed to approximate the Nash equilibrium of the game. Experimental evaluations involve both an event-driven simulator and real-world system, where the proposed MARL load balancing algorithm shows close-to-optimal performance in simulations, and superior results over in-production LBs in the real-world system.

1 Introduction

In cloud data centers (DCs) and distributed networking systems, servers are deployed on infrastructures with multiple processors to provide scalable services [1]. To optimise workload distribution and reduce additional queuing delay, load balancers (LBs) play a significant role in such systems. State-of-the-art network LBs rely on heuristic mechanisms [2–5] under the low-latency and high-throughput constraints of the data plane. However, these heuristics are not adaptive to dynamic environments and require human interventions, which can lead to most painful mistakes in the cloud – mis-configurations. RL approaches have shown performance gains in distributed system and networking problems [6–9], yet applying RL on the network load balancing problem is challenging.

First, unlike traditional workload distribution or task scheduling problem [6, 7], network LBs have limited observations over the system, including task sizes and actual server load states. Being aware of only the number of tasks they have distributed, servers can be overloaded by collided elephant tasks and have degraded quality of service (QoS). Second, to guarantee high service availability in the cloud, multiple LBs are deployed in DCs. Network traffic is split among all LBs. This multi-agent setup makes LBs have only partial observation over the system. Third, modern DCs are based on heterogeneous hardware and elastic infrastructures [10], where server capacities vary. It is challenging to assign correct weights to servers according to their actual processing capacities, and this process conventionally requires human intervention – which can lead to error-prone configurations [3, 5]. Last

*Equal contribution.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

but not least, given the low-latency and high-throughput constraints in the distributed networking setup, the interactive training procedure of RL models and the centralised-training-decentralised-execution (CTDE) scheme [11] can incur additional communication and management overhead.

In this paper, we study the network load balancing problem in multi-agent game theoretical approach, by formulating it as a Markov potential game through specifying the proper reward function, namely variance-based fairness. We propose a distributed Multi-Agent RL (MARL) network load balancing mechanism that is able to exploit asynchronous actions based only on local observations and inferences. Load balancing performance gains are evaluated based on both event-based simulations and real-world experiments[2].

Algorithm 1 LB System Transition Protocol

Initialise server load, $X_j(0) \leftarrow 0, \forall j \in [N]$
for each time step t **do**
  for each LB agent $i \in [M]$ **do**
    Choose action $\alpha_{ij}(t)$ for coming tasks $w_i(t)$
  end for
  for each server j **do**
    Update workload:
    $X_j(t) = X_j(t-1) + \sum_{i=1}^{M} w_i(t)\alpha_{ij}(t) - v_j(t-1)$
  end for
end for

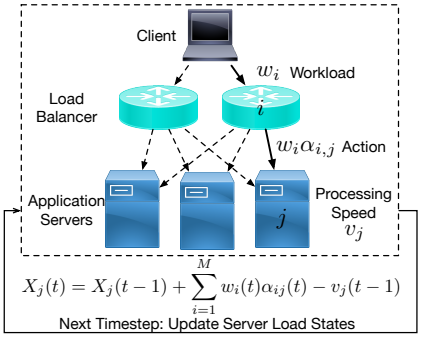


Figure 1: Network load balancing.

2 Related Work

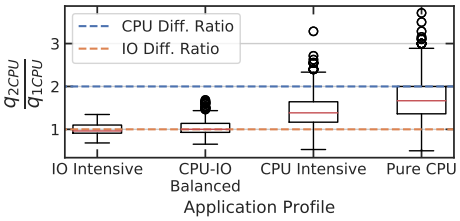
Network Load Balancing Algorithms. The main goal of network LBs is to *fairly* distribute workloads across servers. The system transition protocol of network load balancing system is described in Alg. 1 and depicted in Fig. 1. Existing load balancing algorithms are sensitive to partial observations and inaccurate server weights. Equal-Cost Multi-Path (ECMP) LBs randomly assign servers to new requests [12–14], which makes them agnostic to server load state differences. Weighted-Cost Multi-Path (WCMP) LBs assign weights to servers proportional to their provisioned resources (*e.g.* CPU power) [3, 15–17]. However, the statically assigned weights may not correspond to the actual server processing capacity. As depicted in Fig. 2a, servers with the same IO speed yet different CPU capacities have different actual processing speed when applications have different resource requirements. Active WCMP (AWCMP) is a variant of WCMP and it periodically probe server utilisation information (CPU/memory/IO usage) [5, 18]. However, active probing can cause delayed observations and incur additional control messages, which degrades the performance of distributed networking systems. Local Shortest Queue (LSQ) assigns new requests to the server with the minimal number of ongoing networking connections that are *locally* observed [19, 20]. It does not concern server processing capacity differences. Shortest Expected Delay (SED) derives the "expected delay" as locally observed server queue length divided by statically configured server processing speed [2]. However, LSQ and SED are sensitive to partial observations and misconfigurations. As depicted in Fig. 2b, the QoS performance of each load balancing algorithm degrades from the ideal setup (global observations and accurate server weight configurations) when network traffic is split across multiple LBs or server weights are mis-configured[3], which prevails in real-world cloud DCs.

In this paper, we propose a distributed MARL-based load balancing algorithm that considers dynamically changing queue lengths (*e.g.* sub-ms in modern DC networks [21]), and autonomously adapts to actual server processing capacities, with no additional communications among LB agents or servers.

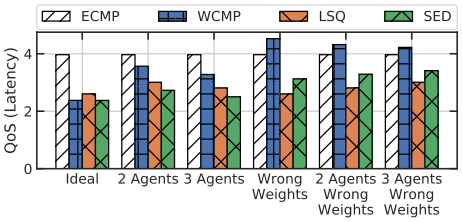
Markov Potential Games. A potential game (PG) [22–25] has a special function called *potential function*, which specifies a property that any individual deviation of the action for one player will change the value of its own and the potential function equivalently. A desirable property of PG is that pure NE always exists and coincides with the maximum of potential function in norm-form setting. Self-play [26] is provably converged for PG. Markov games (MG) is an extension of normal-form

[2]Source code and data of both simulation and real-world experiment are open-sourced at https://github.com/ZhiyuanYaoJ/MARLLB.

[3]The stochastic Markov model of the simulation is detailed in the App. A



(a) It is hard to accurately estimate the actual server processing speeds since it depends on both provisioned resources, and application profiles (App. E.1).



(b) The performance of existing network load balancing algorithms degrades when observation becomes partial with multi-agents and weights are mis-configured.

Figure 2: Existing network load balancing algorithms are sub-optimal under real-world setups.

Table 1: Trade-offs among the probing frequency, measurement quality, and communication overhead.

Probing Frequency (/s)		2.22	2.86	4.00	6.67	20.00
RMSE	CPU (%)	48.33	44.56	39.84	32.65	21.97
	#Job	2.07	1.85	1.61	1.31	0.91
Spearman's Corr.	CPU (%)	0.28	0.40	0.52	0.68	0.85
	#Job	0.47	0.56	0.66	0.77	0.89
Communication Overhead (kbps)	2LB-7server	2.15	2.76	3.86	6.44	9.32
	6LB-20server	18.40	23.66	33.12	55.20	165.60

game to a multi-step sequential setting. A combination of PG and MG yields the Markov potential games (MPG) [27, 28], where pure NE is also proved to exist. Some algorithms [27, 29, 30] lying in the intersection of game theory and reinforcement learning are proposed for MPG. For example, independent nature policy gradient is proved to converge to Nash equilibrium (NE) for MPG [27].

Multi-Agent RL. MARL [31] has been viewed as an important avenue for solving different types of games in recent years. For cooperative settings, a line of work based on joint-value factorisation have been proposed, involving VDN [32], COMA [11], MADDPG [33], and QMIX [34]. For these works, a global reward is assigned to players within the team, but individual policies are optimised to execute individual actions, known as the CTDE setting. MPG satisfies the assumptions of the value decomposition approach, with the well-specified potential function as the joint rewards. However, deploying CTDE RL models in real-world distributed system incurs additional communication latency and management overhead for synchronising agents and aggregating trajectories. These additional management and communication overheads can incur substantial performance degradation – constrained throughput and increased latency – especially in data center networks. As listed in Table 1, when we use active probing to measure server utilisation information, higher probing frequencies give improved measurement quality–in terms of CPU usage and number of on-going jobs on the servers. However, higher probing frequencies also incur increased communication overhead, especially in large-scale data center networks. The detailed experimental setups, as well as both qualitative and quantitative analysis of the impact of communication overhead, are described in Sec. E.2.2. By leveraging the special structure of MPG, independent learning approach can be more efficient due to the decomposition of the joint state and action spaces, which is leveraged in the proposed methods. Methods like MATRPO [35], IPPO [36] follow a fully decentralised setting, but for general cooperative games.

In terms of the distribution fairness, FEN [37] is proposed as a decentralised approach for fair reward distribution in multi-agent systems. They defined the fairness as coefficient of variation and decompose it for each individual agent. Another work [38] proposes a decentralised learning method for fair policies in cooperative games. However, the decentralised learning manner in these methods are not well justified, while in this paper the load balancing problem is formally characterised as a MPG and the effectiveness of distributed training is verified.

3 Methods

3.1 Problem Description

We formulate the load balancing problem into a discrete-time dynamic game with strong distributed and concurrent settings, where no centralised control mechanism exists among agents. We let M

denote the number of LB agents ($[M]$ denotes the set of LB agents $\{1, \ldots, M\}$) and N denote the number of servers ($[N]$ denotes the set of servers $\{1, \ldots, N\}$). At each time step (or round) $t \in H$ in a horizon H of the game, each LB agent i receives a workload $w_i(t) \in W$, where W is the workload distribution, and the LB agent assigns a server to the task using its load balancing policy $\pi_i \in \Pi$, where Π is the load balancing policy profile. At each time-step t, a LB agent i takes an action $a_i(t) = \{a_{ij}(t)\}_{j=1}^N$, according to which the tasks $w_i(t)$ are assigned with distribution $\alpha_i(t)$. $\alpha_{ij}(t)$ is the probability mass of assigning tasks to server j, $\sum_{j=1}^N \alpha_{ij}(t) = 1$. Therefore, at each time step, the workload assigned to server j by the i-th LB is $w_i(t)\alpha_{ij}(t)$. During each time interval, each server j is capable of processing a certain amount of workload v_j based on the property of each server (*e.g.* provisioned resources including CPU, memory, etc.). We have server load state (remaining workload to process) $X_j(T) = \sum_{t=0}^T \max\{0, \sum_{i=1}^M w_i(t)\alpha_{ij}(t) - v_j\} = \max\{0, \sum_{t=0}^T \sum_{i=1}^M w_i(t)\alpha_{ij}(t) - v_j T\} = \sum_{i=1}^M X_{ij}(T)$[4]. Let l_j denote the time for a server j to process all remaining workloads, which is also the potential queuing time for new-coming tasks, $l_j(t) = \frac{X_j(t-1)+\sum_{i=1}^M w_i(t)\alpha_{ij}(t)}{v_j} = \frac{\sum_{i=1}^M X_{ij}(t-1)+w_i(t)\alpha_{ij}(t)}{v_j} = \sum_{i=1}^M l_{ij}(t)$. Then transition from time step t to time step $t+1$ is given in Alg. 1. Reward: $r_i(t) = R(\boldsymbol{l}(t), a_i(t), \delta_i(t))$, where R is the reward function, $\boldsymbol{l}(t) = \sum_{j=1}^N l_j(t) = \sum_{i=1}^M l_i(t)$ denotes the estimated remaining time to process on each server, and $\delta_i(t)$ is a random variable that makes the process stochastic.

Definition 1. *(Makespan) In the selfish load balancing problem, the makespan is defined as:*

$$MS = \max_j(l_j), l_j = \sum_i l_{ij} \tag{1}$$

The network load balancing problem is featured as multi-commodity flow problems and is NP-hard, which makes it hard to solve with trivial algorithmic solution within micro-second level [39]. This problem can be formulated as a constrained optimisation problem for minimizing the makespan over an horizon $t \in [H]$:

$$minimize \sum_{t=h}^{H} \max_j l_j(t) \tag{2}$$

$$s.t. \quad l_j(t) = \frac{\sum_{i=1}^M (X_{ij}(t-1) + w_i(t)\alpha_{ij}(t))}{v_j}, \quad \sum_{i=1}^M w_i(t) \leq \sum_{j=1}^N v_j, \quad w_i, v_j \in (0, +\infty) \tag{3}$$

$$X_{ij}(T) = \sum_{t=0}^T \max\{0, w_i(t)\alpha_{ij}(t) - \frac{v_j}{M}\}, \quad \sum_{j=1}^N \alpha_{ij}(t) = 1, \quad \alpha_{ij} \in [0, 1] \tag{4}$$

In modern realistic network load balancing system, the arrival of network requests is usually unpredictable in both its arriving rate and the expected workload, which introduces large stochasticity into the problem. Moreover, due to the existence of noisy measurements and partial observations, the estimation of makespan can be inaccurate, which indicates the actual server load states or processing capacities are not correctly captured. Instant collisions of elephant workloads or bursts of mouse workloads often happen, which do not indicate server processing capacity thus misleading the observation. To solve this issue, we introduce *fairness* as an alternative of the original objective makespan. Specifically, makespan is estimated on a per-server level, while the estimation of fairness can be decomposed to the LB level, which allows evaluating the individual LB performance without general loss. This is more natural in load balancing system due to the partial observability of LBs.

3.2 Distribution Fairness

We mainly introduce two types of load balancing distribution fairness: (1) variance-based fairness (VBF) and (2) product-based fairness (PBF). It will be proved that optimization over either fairness will be sufficient but not necessary for minimising the makespan.

Definition 2. *(Variance-based Fairness) For a vector of time to finish all remaining jobs $\boldsymbol{l} = [l_1, \ldots, l_N]$ on each server $j \in [N]$, let $\bar{\boldsymbol{l}}(t) = \frac{1}{N}\sum_{j=1}^N \sum_{i=1}^M l_{ij}(t)$, the variance-based fairness for*

[4] $X_{ij}(T) = \sum_{t=0}^T \max\{0, w_i(t)\alpha_{ij}(t) - \frac{v_j}{M}\}$

workload distribution is just the negative sample variance of the job time, which is defined as:

$$F(\boldsymbol{l}) = -\frac{1}{N}\sum_{j=1}^{N}\Big(l_j(t) - \bar{\boldsymbol{l}}(t)\Big)^2 = -\frac{1}{N}\sum_{j=1}^{N} l_j^2(t) + \bar{\boldsymbol{l}}^2(t). \tag{5}$$

VBF defined per LB is: $F_i(\boldsymbol{l}_i) = -\frac{1}{N}\sum_{j=1}^{N} l_{ij}^2(t) + \bar{\boldsymbol{l}}_i^2(t)$*, where* $\bar{\boldsymbol{l}}_i(t) = \frac{1}{N}\sum_{j=1}^{N} l_{ij}(t)$.

Lemma 3. *The VBF for load balancing system satisfies the following property:*

$$F_i^{\pi_i,-\pi_i}(\boldsymbol{l}_i) - F_i^{\tilde{\pi}_i,-\pi_i}(\tilde{\boldsymbol{l}}_i) = F^{\pi_i,-\pi_i}(\boldsymbol{l}) - F^{\tilde{\pi}_i,-\pi_i}(\tilde{\boldsymbol{l}}) \tag{6}$$

This property makes VBF a good choice for the reward function in load balancing tasks. We will see more discussions in later sections. Proof of the lemma is provided in Appendix B.1.

Proposition 4. *Maximising the VBF is sufficient for minimising the makespan, subjective to the load balancing problem constraints (Eq.* (3) *and* (4)*):* $\max F(\boldsymbol{l}) \Rightarrow \min\max_j(l_j)$*. This also holds for per-LB VBF as* $\max F_i(\boldsymbol{l}_i) \Rightarrow \min\max_j(\boldsymbol{l}_i)$.

Definition 5. *(Product-based Fairness [40]) For a vector of time to finish all remaining jobs* $\boldsymbol{l} = [l_1, \ldots, l_N]$ *on each server* $j \in [N]$*, the product-based fairness for workload distribution is defined as:* $F(\boldsymbol{l}) = F([l_1, \ldots, l_N]) = \prod_{j\in[N]} \frac{l_j}{\max(\boldsymbol{l})}$*. PBF defined per LB is:* $F_i(\boldsymbol{l}_i) = F([l_{i1}, \ldots, l_{iN}]) = \prod_{j\in[N]} \frac{l_{ij}}{\max(\boldsymbol{l}_i)}$.

Proposition 6. *Maximising the product-based fairness is sufficient for minimising the makespan, subjective to the load balancing problem constraints (Eq.* (3) *and* (4)*):* $\max F(\boldsymbol{l}) \Rightarrow \min\max(\boldsymbol{l})$.

Proofs of proposition 4 and 6 are in Appendix B.1 andB.2, respectively. From proposition 4 and 6, we know that the two types of fairness can serve as an effective alternative objective for optimising the makespan, which will be leveraged in our proposed MARL method as valid reward functions.

3.3 Game Theory Framework

Markov game is defined as $\mathcal{MG}(H, M, \mathcal{S}, \mathcal{A}_{\times M}, \mathbb{P}, r_{\times M})$, where H is the horizon of the game, M is the number of player in the game, $\mathcal{S}$ is the state space, $\mathcal{A}_{\times M}$ is the joint action space of all players, $\mathcal{A}_i$ is the action space of player i, $\mathbb{P} = \{\mathbb{P}_h\}, h \in [H]$ is a collection of transition probability matrices $\mathbb{P}_h : \mathcal{S} \times \mathcal{A}_{\times M} \to \Pr(\mathcal{S})$, $r_{\times M} = \{r_i | i \in [M]\}, r_i : \mathcal{S} \times \mathcal{A}_{\times M} \to \mathbb{R}$ is the reward function for i-th player given the joint actions. The stochastic policy space for the i-th player in $\mathcal{MG}$ is defined as $\Pi_i : \mathcal{S} \to \Pr(\mathcal{A}_i), \Pi = \{\Pi_i\}, i \in [M]$.

For the Markov game $\mathcal{MG}$, the state value function $V_{i,h}^{\boldsymbol{\pi}} : \mathcal{S} \to \mathbb{R}$ and state-action value function $Q_{i,h}^{\boldsymbol{\pi}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for the i-th player at step h under policy $\boldsymbol{\pi} \in \Pi_{\times M}$ is defined as:

$$V_{i,h}^{\boldsymbol{\pi}}(s) := \mathbb{E}_{\boldsymbol{\pi},\mathbb{P}}\Big[\sum_{h'=h}^{H} r_{i,h'}(s_{h'}, \boldsymbol{a}_{h'})\Big| s_h = s\Big], Q_{i,h}^{\boldsymbol{\pi}}(s, \boldsymbol{a}) := \mathbb{E}_{\boldsymbol{\pi},\mathbb{P}}\Big[\sum_{h'=h}^{H} r_{i,h'}(s_{h'}, \boldsymbol{a}_{h'})\Big| s_h = s, a_h = \boldsymbol{a}\Big]. \tag{7}$$

Definition 7. *(ϵ-approximate Nash equilibrium) Given a Markov game* $\mathcal{MG}(H, M, \mathcal{S}, \mathcal{A}_{\times M}, \mathbb{P}, \Pi_{\times M}, r_{\times M})$*, let* π_{-i} *be the policies of the players except for the* i*-th player, the policies* (π_i^*, π_{-i}^*) *is an* ϵ*-Nash equilibrium if* $\forall i \in [M], \exists \epsilon > 0$*,*

$$V_i^{\pi_i^*,\pi_{-i}^*}(s) \geq V_i^{\pi_i,\pi_{-i}^*}(s) - \epsilon, \forall \pi_i \in \Pi_i. \tag{8}$$

If $\epsilon = 0$*, it is an exact Nash equilibrium.*

Definition 8. *(Markov Potential Game) A Markov game* $\mathcal{M}(H, M, \mathcal{S}, \mathcal{A}_{\times M}, \mathbb{P}, \Pi_{\times M}, r_{\times M})$ *is a Markov potential game (MPG) if* $\forall i \in [M], \pi_i, \tilde{\pi}_i \in \Pi_i, \pi_{-i} \in \Pi_{-i}, s \in \mathcal{S}$*,*

$$V_i^{\pi_i,\pi_{-i}}(s) - V_i^{\tilde{\pi}_i,\pi_{-i}}(s) = \phi^{\pi_i,\pi_{-i}}(s) - \phi^{\tilde{\pi}_i,\pi_{-i}}(s), \tag{9}$$

where $\phi(\cdot)$ *is the potential function independent of the player index.*

Lemma 9. *Pure NE (PNE) always exists for PG, local maximisers of potential function are PNE. PNE also exists for MPG. [22]*

Theorem 10. *Multi-agent load balancing is MPG with the VBF* $F_i(\boldsymbol{l}_i)$ *as the reward* r_i *for each LB agent* $i \in [M]$*, then suppose for* $\forall s \in \mathcal{S}$ *at step* $h \in [H]$*, the potential function is time-cumulative total fairness:* $\phi^{\pi_i,-\pi_i}(s) = \sum_{t=h}^{H} F^{\pi_i,-\pi_i}(\boldsymbol{l}(t))$.

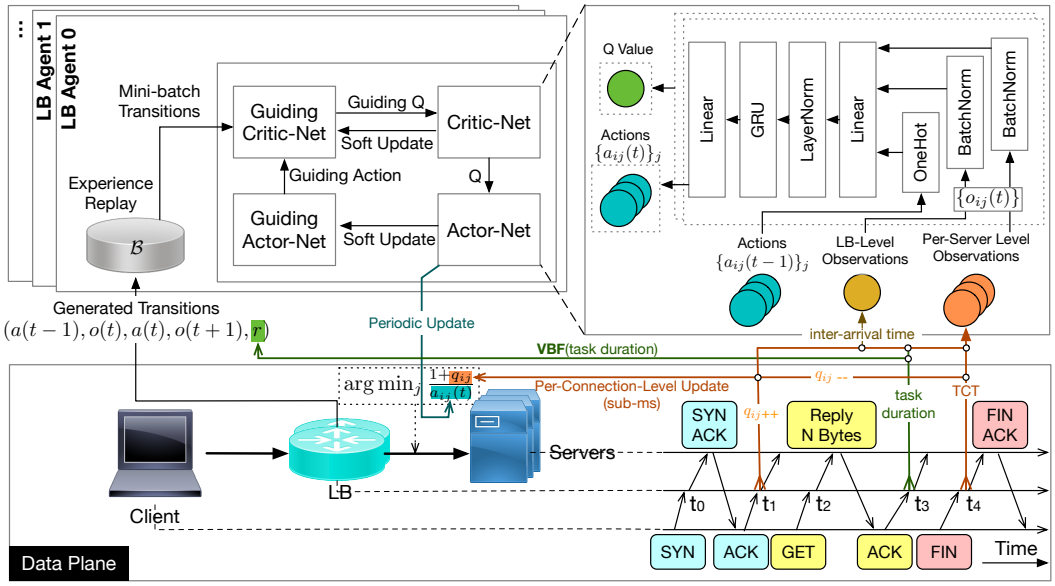


Figure 3: Overview of the proposed distributed MARL framework for network LB.

The proof of the theorem is based on Lemma 3, and it's provided in Appendix B.3. This theorem is essential for establishing our method, since it proves that multi-agent load balancing problem can be formulated as a MPG with the time-cumulative VBF as its potential function. Also, the choice of per-LB VBF as reward function for individual agent is critical for making it MPG, it is easy to verify that PBF cannot guarantee such property. From Lemma 9 we know the maximiser of potential function is the NE of MPG, and from proposition 4 it is known that maximising the VBF gives the sufficient condition for minimising the makespan. Therefore, an effective independent optimisation with respect to the individual reward function specified in the above theorem will lead the minimiser of makespan for load balancing tasks. The effective independent optimisation here means the NE of MPG is achieved.

3.4 Distributed LB Method

With the above analysis, the load balancing problem can be formulated as an episodic version of multi-player partially observable Markov game, which we denote as $\mathcal{POMG}(H, M, \mathcal{S}, \mathcal{O}_{\times M}, \mathbb{O}_{\times M}, \mathcal{A}_{\times M}, \mathbb{P}, r_{\times M})$, where $M, H, \mathcal{S}, \mathcal{A}_{\times M}$ and $\mathbb{P}$ follow the same definitions as in Markov game $\mathcal{MG}$, $\mathcal{O}_{\times M}$ contains the observation space O_i for each player, $\mathbb{O} = \{\mathbb{O}_h\}, h \in [H]$ is a collection of observation emission matrices, $\mathbb{O}_{i,h} : \mathcal{S} \to \Pr(\mathcal{O}_i)$, $r_{\times M} = \{r_i | i \in [M]\}, r_i : \mathcal{O}_i \times \mathcal{A}_{\times M} \to \mathbb{R}$ is the reward function for i-th LB agent given the joint actions. The stochastic policy space for the i-th agent in $\mathcal{POMG}$ is defined as $\Pi_i : \mathcal{O}_i \to \Pr(\mathcal{A}_i)$. As discussed in Sec. 2, the partial observability comes from the fundamental configuration of network LBs in DC networks, which allows LBs to observe only a partial of network traffic and does not give LBs information about the tasks (*e.g.* expected workload) distributed from each LB. The reward functions in our experiments are variants of distribution fairness introduced in Sec. 3.2. The potential functions can be defined accordingly based on the two fairness indices. The overview of the proposed distributed MARL framework is shown in Fig. 3.

In MPG, independent policy gradient allows finding the maximum of the potential function, which is the PNE for the game. This inspires us to leverage the policy optimisation in a decomposed manner, *i.e.*, distributed RL for policy learning of each LB agent. However, due to the partial observability of the system and the challenge of directly estimating the makespan (Eq. (1)), each agent cannot have a direct access to the global potential function. To address this problem, the aforementioned fairness (Sec. 3.2) can be deployed as the reward function for each agent, which makes the value function as a valid alternative for the potential function as an objective. This also transforms the joint objective (makespan or potential) to individual objectives (per LB fairness) for each agent. Proposition 4 and 6 verify that optimising towards these fairness indices is sufficient for minimising the makespan.

Algorithm 2 Distributed LB for MPG

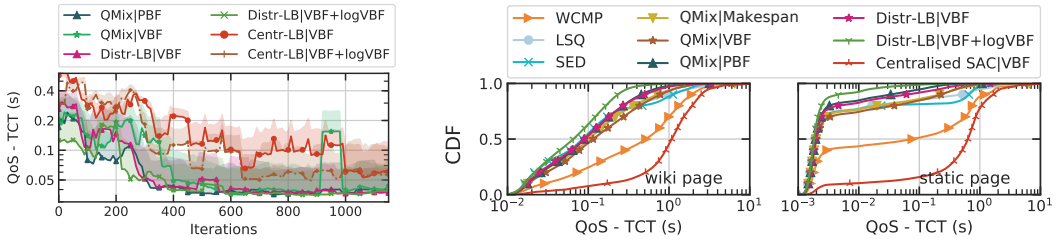
1: **Initialise:**
2: LB policy π_{θ_i} and critic Q_{ϕ_i} networks, replay buffer $\mathcal{B}_i, \forall i \in [M]$;
3: server processing speed function $v_j, \forall j \in [N]$;
4: initial observed instant queue length on server j by the i-th LB: $q_{ij} = 0, \forall i \in [M], j \in [N]$.
5: **while** not converge **do**
6: Reset server load state $X_j(1) \leftarrow 0, \forall j \in [N]$
7: Each LB agent i ($i \in [M]$) receives individual observation $o_i(1)$
8: **for** $t = 1, \ldots, H$ **do**
9: Initialise distributed workload $m_{ij}, w_i(t) \leftarrow 0, i \in [M], j \in [N]$
10: Get actions $a_i(t) \leftarrow \{a_{ij}(t)\}_{j=1}^{N} = \pi_{\theta_i}(o_i(t)), i \in [M]$
11: **for** job $\tilde{w}$ arrived at LB i between timestep $[t, t+1)$ **do**
12: LB i assigns $\tilde{w}$ to server $j = \arg\min_{k \in [N]} \frac{q_{ik}(t)+1}{a_{ik}(t)}$
13: $m_{ij} \leftarrow m_{ij} + \tilde{w}, w_i(t) \leftarrow w_i(t) + \tilde{w}$
14: $\alpha_{ij}(t) \leftarrow \frac{m_{ij}}{w_i(t)}$
15: **end for**
16: **for** each server j **do**
17: Update workload: $X_{ij}(t+1) \leftarrow \max\{X_{ij}(t) + w_i(t)\alpha_{ij}(t) - \frac{v_j}{M}, 0\}$
18: $X_j(t+1) \leftarrow \sum_{i=1}^{M} X_{ij}(t)$
19: **end for**
20: Each agent receives individual reward $r_i(t)$
21: Each agent i collects observation $o_i(t+1), i \in [M]$
22: Update replay buffer: $\mathcal{B}_i = \mathcal{B}_i \bigcup (a_i(t-1), o_i(t), a_i(t), r_i(t), o_i(t+1)), i \in [M]$
23: **end for**
24: Update critics with gradients: $\nabla_{\phi_i} \mathbb{E}_{(o_i, a_i, r_i, o_i') \sim \mathcal{B}_i} \left[\left(Q_{\phi_i}(o_i, a_i) - r_i - \gamma V_{\tilde{\phi}_i}(o_i') \right)^2 \right]$
25: where $V_{\tilde{\phi}_i}(o_i') = \mathbb{E}_{(o_i', a_i') \sim \mathcal{B}_i}[Q_{\tilde{\phi}_i}(o_i', a_i') - \alpha \log \pi_{\theta_i}(a_i'|o_i')], i \in [M]$
26: Update policies with gradients: -$\nabla_{\theta_i} \mathbb{E}_{o_i \sim \mathcal{B}_i}[\mathbb{E}_{a \sim \pi_{\theta_i}}[\alpha \log \pi_{\theta_i}(a_i|o_i) - Q_{\phi_i}(o_i, a_i)]], i \in [M]$
27: **end while**
28: **return** final models of learning agents

Alg. 2 shows the proposed distributed LB for load balancing problem, which is a partially observable MPG. The distributed policy optimisation is based on Soft Actor-Critic (SAC) [41] algorithm, which is a type of maximum-entropy RL method. It optimises the objective $\mathbb{E}[\sum_t \gamma^t r_t + \alpha \mathcal{H}(\pi_\theta)]$, whereas $\mathcal{H}(\cdot)$ is the entropy of the policy π_θ. Specifically, the critic Q network is updated with gradient $\nabla_\phi \mathbb{E}_{o,a} \left[\left(Q_\phi(o,a) - r(o,a) - \gamma \mathbb{E}_{o'}[V_{\tilde{\phi}}(o')] \right)^2 \right]$, where $V_{\tilde{\phi}}(o') = \mathbb{E}_{a'}[Q_{\tilde{\phi}}(o', a') - \alpha \log \pi_\theta(a'|o')]$ and $Q_{\tilde{\phi}}$ is the target Q network; the actor policy π_θ is updated with the gradient $\nabla_\theta \mathbb{E}_o[\mathbb{E}_{a \sim \pi_\theta}[\alpha \log \pi_\theta(a|o) - Q_\phi(o,a)]]$. Other key elements of RL methods involve the observation, action and reward function, which are detailed as following.

Observation. Each LB agent partially observes over the traffic that traverses through itself, including per-server-level and LB-level measurements. For each LB, per-server-level observations consist of – for each server – the number of on going tasks, and sampled task duration and task completion time (TCT). Specifically, in Alg. 2 line 12-14, w_i is the coming workload on servers assigned by i-th LB, and it is not observable for LB. $q_{ik} + 1$ is the locally observed number of tasks on k-th server by i-th LB, due to the real-world constraints of limited observability at the Transport layer. The "+1" is for taking into account the new-coming task. Observations of task duration and TCT samples, along with LB-level measurements which sample the task inter-arrival time as an indication of overall system load state, are reduced to 5 scalars – *i.e.* average, 90th-percentile, standard deviation, discounted average and weighted discounted average[5] – as inputs for LB agents.

Action. To bridge the different timing constraints between the control plane and data plane, each LB agent assigns the j-th server to newly arrived tasks using the ratio of two factors, $\arg\min_{k \in [N]} \frac{q_{ik}+1}{a_{ik}}$, where the number of on-going tasks q_{ik} helps track dynamic system server occupation at per-connection level – which allows making load balancing decision at μs-level speed – and a_{ik} is

[5]Discounted average weights are computed as $0.9^{t'-t}$, where t is the sample timestamp and t' is the moment of calculating the reduced scalar.



(a) Learning curves. (b) CDF of TCT.

Figure 4: Experimental results show that the proposed distributed RL framework using proposed VBF as rewards converges and effectively achieves better load balancing performance (lower TCT and better QoS) than existing LB algorithms and CTDE RL algorithms.

Table 2: Comparison of average QoS (s) in moderate-scale real-world network setup.

Method		Period III (758.787 queries/s)		Period IV (784.522 queries/s)	
		Wiki	Static	Wiki	Static
WCMP		0.412 ± 0.101	0.134 ± 0.059	0.834 ± 0.323	0.492 ± 0.276
LSQ		0.620 ± 0.442	0.339 ± 0.316	0.357 ± 0.373	0.173 ± 0.299
SED		0.215 ± 0.210	0.051 ± 0.081	0.346 ± 0.496	0.169 ± 0.330
RLB-SAC [40]	Jain	0.193 ± 0.073	0.026 ± 0.022	0.204 ± 0.084	0.039 ± 0.047
	G	0.149 ± 0.049	0.015 ± 0.011	0.155 ± 0.052	0.011 ± 0.011
QMix-LB	MS	0.217 ± 0.157	0.048 ± 0.069	0.263 ± 0.202	0.073 ± 0.092
	VBF	0.141 ± 0.025	0.008 ± 0.004	0.286 ± 0.162	0.068 ± 0.066
	PBF	0.211 ± 0.153	0.047 ± 0.078	0.181 ± 0.042	0.018 ± 0.009
Distr-LB (this paper)	VBF	0.159 ± 0.054	0.017 ± 0.009	0.196 ± 0.091	0.032 ± 0.033
	VBF+logVBF	$\mathbf{0.108 \pm 0.022}$	$\mathbf{0.004 \pm 0.001}$	$\mathbf{0.104 \pm 0.013}$	$\mathbf{0.006 \pm 0.003}$
Centr-LB	VBF	1.068 ± 0.386	0.570 ± 0.378	1.378 ± 0.377	0.867 ± 0.350
	VBF+logVBF	0.759 ± 0.254	0.306 ± 0.222	1.013 ± 0.168	0.520 ± 0.167

the periodically updated RL-inferred server processing speed. As in line 14 of Alg. 2, $\alpha_{ij}(t)$ is a statistical estimation of workload assignment distribution at time interval $[t, t+1)$.

Reward. The individual reward for distributed MPG LB is chosen as the VBF (as Def. 2) of the discounted average of sampled task duration measured on each LB agent, such that the LB group jointly optimise towards the potential function defined in Eq. (10). Task duration information is gathered as the time interval between the end of connection initialisation (*e.g.* 3-way handshake for TCP traffic) and the acknowledgement to the first data packet (*e.g.* the first ACK packet for TCP traffic). Given the limited and partial observability of LB agents, task duration information approximates the remaining workload $\boldsymbol{l}$ by measuring the queuing and processing delay for new-coming tasks on each server. This PBF- and MS-based rewards are also implemented for CTDE MARL algorithm as a comparison.

Model. The architecture of the proposed RL framework is depicted in Fig. 3. Each LB agent consists of a replay buffer, and a pair of actor-critic networks, whose architecture is depicted on the top right. There is also a pair of guiding actor-critic networks, with the same network architectures but updated in a delayed and soft manner. Each LB agent takes observations $o_i(t)$ extracted from the data plane (*e.g.* numbers of ongoing tasks $\{q_{ij}\}$, task duration, TCT) and actions from previous timestep $a_i(t-1)$ as inputs, and periodically generates new actions $a_i(t)$, which is used to update the server assignment function $\arg\min_{j\in[N]} \frac{q_{ij}+1}{a_{ij}}$ in the data plane. The gated recurrent units (GRU) [42] are applied for all agents to leverage the sequential history information for handling partial observability.

4 Evaluation

We developed (i) an event-based simulator (App. C.1) to study the distance between the NE achieved by the proposed algorithm and the NE achieved by the theoretical optimal load balancing policy (with perfect observation), and (ii) a realistic testbed (App. C.2) on physical servers in a DC network providing Apache web services, with real-world network traffic [43], to evaluate the real-world performance of the proposed algorithm, in comparison with in-production state-of-the-art LB [3].

Table 3: Comparison of average QoS (s) in moderate-scale simulator for different types of applications.

		50%-CPU+50%-IO	75%-CPU+25%-IO	100%-CPU
Oracle		6.437 ± 1.006	1.469 ± 0.102	1.291 ± 0.075
QMix-LB	PBF	10.230 ± 0.108	1.828 ± 0.054	2.200 ± 0.288
	VBF	10.936 ± 0.470	2.023 ± 0.255	2.125 ± 0.074
Distr-LB (this paper)	VBF	10.335 ± 0.362	$\mathbf{1.695 \pm 0.104}$	$\mathbf{1.643 \pm 0.016}$
	VBF+logVBF	$\mathbf{8.797 \pm 0.459}$	1.873 ± 0.328	2.004 ± 0.042

Table 4: Comparison of average QoS (s) in large-scale real-world network setup.

Method		Period I (2022.855 queries/s)		Period II (2071.129 queries/s)	
		Wiki	Static	Wiki	Static
WCMP		0.473 ± 0.102	0.194 ± 0.090	0.460 ± 0.241	0.239 ± 0.212
LSQ		0.266 ± 0.127	0.063 ± 0.065	0.218 ± 0.246	0.082 ± 0.152
SED		0.169 ± 0.062	0.020 ± 0.025	0.166 ± 0.141	0.050 ± 0.070
RLB-SAC-G [40]		0.182 ± 0.049	0.013 ± 0.009	0.111 ± 0.029	0.010 ± 0.009
QMix-LB	VBF	0.181 ± 0.062	0.019 ± 0.020	0.188 ± 0.147	0.052 ± 0.075
	PBF	0.210 ± 0.041	0.013 ± 0.006	0.104 ± 0.009	0.005 ± 0.003
Distr-LB (this paper)	VBF	0.228 ± 0.055	0.019 ± 0.011	0.174 ± 0.102	0.035 ± 0.039
	VBF+logVBF	$\mathbf{0.161 \pm 0.033}$	$\mathbf{0.008 \pm 0.003}$	$\mathbf{0.094 \pm 0.015}$	$\mathbf{0.004 \pm 0.001}$

Moderate-Scale Real-World Testbed: As depicted in Fig. 4a, in a moderate-scale real-world DC network setup with 2 LB agents and 7 servers, after 120 episodes of training, the proposed distributed LB (Distr-LB) algorithm is able to learn from the environment based on VBF as rewards, and it converges to offer better QoS than QMix. Centralised RL agent (Centr-LB) has difficulties to learn within 120 episodes because of the increased state and action space. An empirical finding is that, by adding a log term to the VBF-based reward for Distr-LB, we help LB agents to become more sensitive to close-to-0 VBF during training ($\nabla_x \log f(x) > \nabla_x f(x)$ when $f(x) < 1$), therefore achieving better load balancing performance. As depicted in Fig. 4b, when comparing with in-production LB algorithms (WCMP, LSQ, SED), Distr-LB shows clear performance gains and reduced TCT for both types of web pages – Wikipedia pages require to query SQL databases thus they are more CPU-intensive, while static pages are IO-intensive. The comparison of average TCT using different LB algorithms is shown in Table 2 (99th percentile TCT in Table 12). The proposed Distr-LB also shows superior performance than the RL-based solution (RLB-SAC) [40] because of (i) a well designed MARL framework, and (ii) the use of recurrent neural network to handle load balancing problem as a sequential problem.

NE Gap Evaluation with Simulation: To evaluate the gap between the performance of Distr-LB and the theoretical optimal policy, we implement in the simulator an Oracle LB, which has perfect observation (inaccessible in real world) over the system and minimises makespan for each load balancing decision. Table 3 shows that, for different types of applications, Distr-LB is able to achieve closer-to-optimal performance than QMix. As the simulator is implemented based on the load balancing model formulated in this paper, our theoretical analysis can be directly applied, and VBF – as a potential function – helps independent cooperative LB agents to achieve good performance. The additional log term shows empirical performance gains in real-world system, yet it is not necessarily the case in these simulation results. On one hand, the generated traffic of tasks in the simulation has higher expected workload ($>$ 1s mean and stddev), while the log terms is more sensitive to close-to-0 variances, which is the case in real-world experimental setups. On the other hand, though the simulator models the formulated LB problem, it fails to captures the complexity in the real-world system – *e.g.* Apache backlog, multi-processing optimisation, context switching, multi-level cache, network queues etc. For instance, batch processing [44] helps reduce cache and instruction misses, yet yields similar processing time for different tasks, thus the variance of task processing delay decreases and becomes closer to 0 in real-world system. The additional log term exaggerates the low variance differences to better evaluate load balancing decisions. More detailed description about the simulator implementation can be found in App C.1 and ablation study on reward engineering is presented in App E.2.1.

Large-Scale Real-World Testbed: To evaluate the performance of Distr-LB in large-scale DC networks in real world, we scale up the real-world testbed to have 6 LB agents and 20 servers and apply heavier network traffic ($>$ 2000 queries/s) to evaluate the performance of the LB algorithms that achieved the best performance in moderate scale setups, in comparison with in-production LB algorithms. The test results after 200 episodes of training are shown in Table 4, where Distr-LB

Table 5: Comparison of 99-th percentile QoS (s) of Wiki pages under different traffic rates using large-scale real-world setup.

Method		Traffic Rate (queries/s)								
		731.534	1097.3	1463.067	1828.834	2194.601	2377.484	2560.368	2743.251	2926.135
LSQ		0.175 ±0.015	0.212 ±0.025	0.249 ±0.043	0.342 ±0.121	0.827 ±0.572	2.103 ±0.654	10.662 ±2.557	17.656 ±0.714	17.999 ±0.253
SED		0.201 ±0.022	0.261 ±0.079	0.322 ±0.099	0.360 ±0.088	0.618 ±0.268	2.175 ±1.328	11.444 ±3.861	22.086 ±4.892	22.727 ±5.632
Distr-LB (this paper)	VBF	**0.160 ±0.010**	**0.205 ±0.036**	**0.248 ±0.086**	**0.284 ±0.113**	0.567 ±0.306	**1.276 ±0.647**	7.005 ±1.147	10.560 ±1.042	15.745 ±0.254
	VBF+logVBF	0.161 ±0.008	0.216 ±0.052	0.249 ±0.068	0.348 ±0.122	**0.439 ±0.121**	1.533 ±0.670	**4.427 ±0.443**	**9.391 ±0.329**	**15.347 ±0.572**

Table 6: Comparison of 99-th percentile QoS (s) of static pages under different traffic rates using large-scale real-world setup.

Method		Traffic Rate (queries/s)								
		731.534	1097.3	1463.067	1828.834	2194.601	2377.484	2560.368	2743.251	2926.135
LSQ		0.014 ±0.001	0.015 ±0.000	0.015 ±0.000	0.018 ±0.003	0.217 ±0.305	0.856 ±0.554	11.066 ±3.095	16.874 ±0.391	17.155 ±0.217
SED		0.014 ±0.000	0.015 ±0.000	0.016 ±0.001	0.018 ±0.001	0.071 ±0.066	1.252 ±1.489	11.272 ±3.975	21.941 ±5.970	20.708 ±5.423
Distr-LB (this paper)	VBF	0.014 ±0.000	0.015 ±0.000	0.016 ±0.001	**0.017 ±0.000**	**0.041 ±0.025**	**0.338 ±0.364**	6.670 ±1.152	9.743 ±0.863	15.506 ±0.056
	VBF+logVBF	0.014 ±0.000	0.015 ±0.001	0.016 ±0.000	0.018 ±0.002	0.072 ±0.087	0.465 ±0.403	**3.970 ±0.545**	**8.782 ±0.187**	**15.095 ±0.497**

achieves the best performance in all cases. QMix also outperforms in-production LB algorithms. But as a CTDE algorithm, similar to the Centr-LB, it requires agents to communicate their trajectories, which – after 200 episodes of training – become 221MiB communication overhead at the end of each episode (episodic training), whereas 95%-percentile per-destination-rack flow rate is less than 1MiB/s [45].

Scaling Experiments: Using the same large-scale real-world testbed with 6 LB agents and 20 servers, we conduct scaling experiments by applying network traces with different traffic rates, comparing 4 LB methods with the best performances. The 99-th percentile QoS for both Wiki and static pages are shown in Table 5, 6. As listed in Table 5 and 6, under low traffic rates, when servers are all under utilised, the advantage of our proposed Distr-LB is not obvious because all resources are over-provisioned. With the increase of traffic rates (till servers are 100% saturated), our methods outperforms the best classical LB methods. More in-depth discussion and analaysis over the average job completion time for both types of pages in these scaling experiments are shown in Table 14 and 15 in App. E.2.2).

More details regarding the real-world DC testbed implementation is in App. C.2, training details are in App. D, complete evaluation results (both moderate-scale and large-scale) are in App. E and ablation studies – *e.g.* communication overhead of CTDE and centralised RL in real-world system, robustness of MARL algorithms in dynamic DC network environments – can be found in App. E.2.

5 Conclusion and Future Work

This paper proposes a distributed MARL approach for multi-agent load balancing problem, based on Markov potential game formulation. The proposed variance-based fairness for individual LB agent is critical for this formulation. Through this setting, the redundant communication overhead among LB agents is removed, thus improving the overall training and deployment efficiency in real-world systems, with the local observations only. Under such formulation, the effectiveness of our proposed distributed LB algorithm together with the proposed fairness are both theoretically justified and experimentally verified. It demonstrates a performance gain over another commonly applied fairness as well as centralised training methods like QMIX or centralised RL agent, in both simulation and real-world tests with different scales.

References

[1] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*, pages 195–216. Springer, 2017.

[2] The Linux Virtual Server Project - Linux Server Cluster for Load Balancing. `http://www.linuxvirtualserver.org/`.

[3] Daniel E Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. In *NSDI*, pages 523–535, 2016.

[4] Yoann Desmouceaux, Pierre Pfister, Jérôme Tollet, Mark Townsley, and Thomas Clausen. 6lb: Scalable and application-aware load balancing with segment routing. *IEEE/ACM Transactions on Networking*, 26(2):819–834, 2018.

[5] Ashkan Aghdai, Michael I-C Wang, Yang Xu, Charles H-P Wen, and H Jonathan Chao. In-network congestion-aware load balancing at transport layer. *arXiv preprint arXiv:1811.09731*, 2018.

[6] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 191–205. ACM, 2018.

[7] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. *arXiv preprint arXiv:1810.01963*, 2018.

[8] Yue Xu, Wenjun Xu, Zhi Wang, Jiaru Lin, and Shuguang Cui. Load balancing for ultra-dense networks: A deep reinforcement learning based approach. *IEEE Internet of Things Journal*, 6(6):9399–9412, Dec 2019. arXiv: 1906.00767.

[9] Viswanath Sivakumar, Tim Rocktäschel, Alexander H Miller, Heinrich Küttler, Nantas Nardelli, Mike Rabbat, Joelle Pineau, and Sebastian Riedel. Mvfst-rl: An asynchronous rl framework for congestion control with delayed actions. *arXiv preprint arXiv:1910.04054*, 2019.

[10] Adithya Kumar, Iyswarya Narayanan, Timothy Zhu, and Anand Sivasubramaniam. The fast and the frugal: Tail latency aware provisioning for coping with load variations. In *Proceedings of The Web Conference 2020*, pages 314–326, 2020.

[11] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[12] Theo Julienne. GLB: GitHub's open source load balancer, 2018.

[13] João Taveira Araújo, Lorenzo Saino, Lennert Buytenhek, and Raul Landa. Balancing on the edge: Transport affinity without network state. page 111–124, 2018.

[14] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28, 2017.

[15] Facebook Incubator, Apr 2020.

[16] Shouqian Shi, Ye Yu, Minghao Xie, Xin Li, Xiaozhou Li, Ying Zhang, and Chen Qian. Concury: A fast and light-weight software cloud load balancer. page 14, 2020.

[17] Reuven Cohen, Matty Kadosh, Alan Lo, and Qasem Sayah. Lb scalability: Achieving the right balance between being stateful and stateless. *IEEE/ACM Transactions on Networking*, 2021.

[18] Ashkan Aghdai, Cing-Yu Chu, Yang Xu, David Dai, Jun Xu, and Jonathan Chao. Spotlight: Scalable transport layer load balancing for data center networks. *IEEE Transactions on Cloud Computing*, 2020.

[19] Guy Goren, Shay Vargaftik, and Yoram Moses. Distributed dispatching in the parallel server model. *arXiv:2008.00793 [cs]*, Aug 2020. arXiv: 2008.00793.

[20] Tom Barbette, Chen Tang, Haoran Yao, Dejan Kostić, Gerald Q Maguire Jr, Panagiotis Papadimitratos, and Marco Chiesa. A high-speed load-balancer design with guaranteed per-connection-consistency. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 667–683, 2020.

[21] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 139–152, 2015.

[22] Dov Monderer and Lloyd S Shapley. Potential games. *Games and economic behavior*, 14(1):124–143, 1996.

[23] William H Sandholm. Potential games with continuous player sets. *Journal of Economic theory*, 97(1):81–108, 2001.

[24] Jason R Marden, Gürdal Arslan, and Jeff S Shamma. Cooperative control and potential games. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(6):1393–1407, 2009.

[25] Ozan Candogan, Ishai Menache, Asuman Ozdaglar, and Pablo A Parrilo. Flows and decompositions of games: Harmonic and potential games. *Mathematics of Operations Research*, 36(3):474–503, 2011.

[26] Drew Fudenberg, Fudenberg Drew, David K Levine, and David K Levine. *The theory of learning in games*, volume 2. MIT press, 1998.

[27] Roy Fox, Stephen M Mcaleer, Will Overman, and Ioannis Panageas. Independent natural policy gradient always converges in markov potential games. In *International Conference on Artificial Intelligence and Statistics*, pages 4414–4425. PMLR, 2022.

[28] Stefanos Leonardos, Will Overman, Ioannis Panageas, and Georgios Piliouras. Global convergence of multi-agent policy gradient in markov potential games. *arXiv preprint arXiv:2106.01969*, 2021.

[29] Sergio Valcarcel Macua, Javier Zazo, and Santiago Zazo. Learning parametric closed-loop policies for markov potential games. *arXiv preprint arXiv:1802.00899*, 2018.

[30] David H Mguni, Yutong Wu, Yali Du, Yaodong Yang, Ziyi Wang, Minne Li, Ying Wen, Joel Jennings, and Jun Wang. Learning in nonzero-sum stochastic games with potentials. In *International Conference on Machine Learning*, pages 7688–7699. PMLR, 2021.

[31] Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective. *arXiv preprint arXiv:2011.00583*, 2020.

[32] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[33] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[34] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[35] Hepeng Li and Haibo He. Multi-agent trust region policy optimization. *arXiv preprint arXiv:2010.07916*, 2020.

[36] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[37] Jiechuan Jiang and Zongqing Lu. Learning fairness in multi-agent systems. *Advances in Neural Information Processing Systems*, 32, 2019.

[38] Matthieu Zimmer, Claire Glanois, Umer Siddique, and Paul Weng. Learning fair policies in decentralized cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 12967–12978. PMLR, 2021.

[39] Siddhartha Sen, David Shue, Sunghwan Ihm, and Michael J Freedman. Scalable, optimal flow routing in datacenters via local link balancing. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 151–162, 2013.

[40] Zhiyuan Yao, Zihan Ding, and Thomas Clausen. Reinforced cooperative load balancing in data center. *arXiv preprint arXiv:2201.11727*, 2022.

[41] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[42] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[43] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009.

[44] The Fast Data Project (fd.io). Vector Packet Processing (VPP). `https://wiki.fd.io/view/VPP/`.

[45] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 123–137. ACM, 2015. event-place: London, United Kingdom.

[46] Mohammed Shabbir Ali, Pierre Coucheney, and Marceau Coupechoux. Reinforcement learning algorithm for load balancing in self-organizing networks, 2019.

[47] Apache Hadoop. Apache hadoop. *URL http://hadoop. apache. org*, 2011.

[48] Apache Spark. Apache spark. *Retrieved January*, 17:2018, 2018.

[49] Erik-Jan van Baaren. Wikibench: A distributed, wikipedia based web application benchmark. *Master's thesis, VU University Amsterdam*, 2009.

[50] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, et al. Ananta: Cloud scale load balancing. *ACM SIGCOMM Computer Communication Review*, 43(4):207–218, 2013.

[51] Vladimir Olteanu, Alexandru Agache, Andrei Voinescu, and Costin Raiciu. Stateless datacenter load-balancing with beamer. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 125–139, 2018.

[52] Rohan Gandhi, Hongqiang Harry Liu, Y Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. Duet: Cloud scale load balancing with hardware and software. *ACM SIGCOMM Computer Communication Review*, 44(4):27–38, 2015.

[53] Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Efficient communication in multi-agent reinforcement learning via variance based control. *Advances in Neural Information Processing Systems*, 32, 2019.

[54] Ziluo Ding, Tiejun Huang, and Zongqing Lu. Learning individually inferred communication for multi-agent cooperation. *Advances in Neural Information Processing Systems*, 33:22069–22079, 2020.

[55] Jiao Zhang, F Richard Yu, Shuo Wang, Tao Huang, Zengyi Liu, and Yunjie Liu. Load balancing in data center networks: A survey. *IEEE Communications Surveys & Tutorials*, 20(3):2324–2352, 2018.

[56] Rohan Gandhi, Hongqiang Harry Liu, Y Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. Duet: Cloud scale load balancing with hardware and software. *ACM SIGCOMM Computer Communication Review*, 44(4):27–38, 2014.

[57] Noa Zilberman, Yury Audzevich, G Adam Covington, and Andrew W Moore. Netfpga sume: Toward 100 gbps as research commodity. *IEEE micro*, 34(5):32–41, 2014.

[58] Yong Li and Min Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, 2015.

[59] ETSI. Open Source MANO (OSM) Project. `https://osm.etsi.org/`, 2019.

[60] OPNFV. Open Platform for NFV (OPNFV) Project Portal. `https://www.opnfv.org/`, 2019.

[61] OpenStack. OpenStack Project Portal. `https://www.openstack.org/`, 2019.

[62] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service*, pages 1–10, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [No] To our knowledge, the work has no potential negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] The proofs are provided in Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The source code to reproduce both simulation and real-world experimental results has been open-sourced. The URL will be provided on acceptance.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] The error bars are replaced by cumulative distribution function (CDF).
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] This work is mainly based on simulation and the information of computational resources are not relevant.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A] This work does not use existing assets.
   (b) Did you mention the license of the assets? [N/A] This work does not use existing assets.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] This work does not use existing assets.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] This work does not use existing assets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] This work does not use existing assets.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] It does not apply to this work.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] It does not apply to this work.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] It does not apply to this work.

Appendix

A A Stochastic Markov Model of a 2-Server Load Balancing Problem

The simulation results of Fig. 2b is based on a basic load balancing setup of 2 servers with different processing capacities $\frac{v_1}{v_2} = 2$ (*i.e.* server 1 is 2x faster than server 2). Each server has a queue of size Q, such that $0 \leq l_1, l_2 \leq Q$. Traffic arrivals and departures are modeled as Poisson processes with rates λ (observed traffic), γ (unobserved traffic), and v_1, v_2. With sufficiently short timeslots, it can be assumed that only one arrival or departure (at most) happen at a given timeslot (*i.e.* $\sum_{i=1}^{2}(\lambda_i + \gamma_i + v_i) \leq 1$); the system is then Markovian with the state (l_1, l_2), departure rates (μ_1, μ_2), and arrival rates $(\lambda_1, \lambda_2, \gamma_1, \gamma_2)$. For simplicity and stability, the system works at *nominal* capacity (*i.e.* $\lambda + \gamma = v$). With $q_i(n)_{l_i}$ denoting the probability (or probability density function), of server q_i to have a queue length of l_i at time-step n, the transition of server occupations between two time-steps can be described as, for $0 < l_i < Q$ (corner cases are treated accordingly):

$$q_i(n)_{l_i} - q_i(n-1)_{l_i} = (\lambda_i + \gamma_i) \cdot q_i(n-1)_{l_i - 1} + v_i \cdot s_i(n-1)_{l_i+1} - (\lambda_i + \gamma_i + v_i) \cdot q_i(n-1)_{l_i}.$$

The QoS performance of each load balancing algorithm in Fig. 2b is measured as the weighted service duration of a connection ($\sum_{i \in \{1,2\}} \frac{l_i}{l_1 + l_2} \frac{l_i}{\mu_i}$), under different configurations. When the LB has accurate observations and configurations (observing 100% traffic – *i.e.* $\gamma = 0$ – and assigning server weights based on actual processing speeds $\frac{w_1}{w_2} = \frac{v_1}{v_2} = 2$), WCMP and SED have the best performance. When the LB observes only partial network traffic ($50\% - Q$ and $33\% - Q$ corresponds to $\gamma = \lambda$, $\gamma = 2 * \lambda$, respectively) and the rest of the network traffic is uniformly split between the two servers ($\gamma_1 = \gamma_2$), LSQ and SED outperform WCMP, which is agnostic to instant server occupancy. However, partial traffic observation also degrades the performance of LSQ and SED. When LBs have inaccurate server weights ($\sim W$ *i.e.* in case of mis-configuration, $\frac{w_1}{w_2} = \frac{1}{2}$, while $\frac{\mu_1}{\mu_2} = 2$), WCMP and SED exhibit degraded performance even when the LB agent sees all the traffic ($\gamma = 0$). Taking both server queue lengths and processing speeds into account, SED makes more informed load balancing decisions, yet its performance risks being degraded by both partial observations on server queue lengths and inaccurate server weights.

B Analysis of Distribution Fairness

B.1 Analysis of VBF

Lemma 11. *The VBF for load balancing system satisfies the following property:*

$$F_i^{\pi_i, -\pi_i}(\boldsymbol{l}_i) - F_i^{\tilde{\pi}_i, -\pi_i}(\tilde{\boldsymbol{l}}_i) = F^{\pi_i, -\pi_i}(\boldsymbol{l}) - F^{\tilde{\pi}_i, -\pi_i}(\tilde{\boldsymbol{l}}) \tag{10}$$

Proof. From the definition of the variance-based fairness (as Def. 2) we have the following for $\forall i \in [M], j \in [N]$,

$$F^{\pi_i, -\pi_i}(\boldsymbol{l}) = -\frac{1}{N} \sum_{j=1}^{N} (l_j - \bar{\boldsymbol{l}})^2 \tag{11}$$

$$F_i^{\pi_i, -\pi_i}(\boldsymbol{l}_i) = -\frac{1}{N} \sum_{j=1}^{N} (l_{ij} - \overline{l_i})^2 \quad (\overline{l_i} = \frac{1}{N} \sum_{j=1}^{N} l_{ij}) \tag{12}$$

By indexing the agent i as the one to change its strategy and slightly abusing notation, denote $l_j = l_{ij} + l_{-ij}$, where $l_{-ij} = \sum_{k \neq i} l_{kj}$.

$$F^{\pi_i,-\pi_i}(\boldsymbol{l}) = -\frac{1}{N}\sum_{j=1}^{N}(l_{ij} + l_{-ij} - \overline{(l_i + l_{-i})})^2 \quad (\text{where } \overline{(l_i + l_{-i})} = \frac{1}{N}\sum_j (l_{ij} + l_{-ij})) \tag{13}$$

$$= -\frac{1}{N}\sum_{j=1}^{N}[l_{ij} + l_{-ij} - (\bar{l}_i + \bar{l}_{-i})]^2 \tag{14}$$

$$= -\frac{1}{N}\sum_{j=1}^{N}[(l_{ij} - \bar{l}_i)^2 + (l_{-ij} - \bar{l}_{-i})^2 - 2(l_{ij} - \bar{l}_i)(l_{-ij} - \bar{l}_{-i})] \tag{15}$$

$$= -\frac{1}{N}\sum_{j=1}^{N}(l_{ij} - \overline{l_i})^2 - \frac{1}{N}\sum_{j=1}^{N}[(l_{-ij} - \bar{l}_{-i})^2 - \frac{2}{N}\sum_{j=1}^{N}(l_{ij} - \bar{l}_i)(l_{-ij} - \bar{l}_{-i})] \tag{16}$$

$$= F_i^{\pi_i,-\pi_i}(\boldsymbol{l}_i) - \frac{1}{N}\sum_{j=1}^{N}(l_{-ij} - \bar{l}_{-i})^2 \quad (\sum_{j=1}^{N}(l_{ij} - \bar{l}_i) = 0) \tag{17}$$

where the second term is a common term not depend on the changing policy π_i. Therefore, the second term will be cancelled in $F^{\pi_i,-\pi_i}(\boldsymbol{l}) - F^{\tilde{\pi}_i,-\pi_i}(\tilde{\boldsymbol{l}}) = F_i^{\pi_i,-\pi_i}(\boldsymbol{l}_i) - F_i^{\tilde{\pi}_i,-\pi_i}(\tilde{\boldsymbol{l}}_i)$, thus finishes the proof. □

Proposition 12. *Maximising the VBF is sufficient for minimising the makespan, subjective to the load balancing problem constraints (Eq.* (3) *and* (4)*):*

$$\max F(\boldsymbol{l}) \Rightarrow \min \max_j (l_j) \tag{18}$$

this also holds for per-LB VBF as $\max F_i(\boldsymbol{l}_i) \Rightarrow \min \max_j(\boldsymbol{l}_i)$.

Proof. Given the stability constraint in Eq. (3) $\sum_{i=1}^{M} w_i(t) \leq \sum_{j=1}^{N} v_j$, we denote the total amount of workload in the system $C = \sum_{j=1}^{N} l_j$, and $l_k = \max_{j\in[N]} l_j$. Based on the constraint in Eq. (4), we have $C \geq 0$, $l_j(t) \geq 0$.

$$\max F(\boldsymbol{l}) \Leftrightarrow \min -F(\boldsymbol{l}) \tag{19}$$

$$-F(\boldsymbol{l}) = \frac{1}{N}\sum_{j=1}^{N}((l_j) - \bar{\boldsymbol{l}})^2 \tag{20}$$

$$= \frac{1}{N}\sum_{j=1}^{N}(l_j - \frac{C}{N})^2 \tag{21}$$

$$= \frac{1}{N}\sum_{j=1}^{N} l_j^2 - \frac{2C}{N^2}\sum_{j=1}^{N} l_j + \frac{C^2}{N^2} \tag{22}$$

$$= \frac{1}{N}\sum_{j=1}^{N} l_j^2 - \frac{C^2}{N^2} \tag{23}$$

$$\leq [(\max_j l_j)^2 - \frac{C^2}{N^2}] \quad (\text{by means inequality}) \tag{24}$$

with the equivalence achieved when $l_j = l_k, \forall j \neq k, j \in [N]$ holds. Therefore,

$$\max F(\boldsymbol{l}) \Rightarrow \min(l_k)^2 - \frac{C^2}{N^2} \tag{25}$$

$$\Leftrightarrow \min l_k \tag{26}$$

$$\Leftrightarrow \min \max_{j\in[n]} l_j \tag{27}$$

and the condition is sufficient but not necessary because $\min(l_k)^2 - \frac{C^2}{N^2}$ is essentially minimizing the upper bound of $-F(\boldsymbol{l})$. □

B.2 Analysis of PBF

Proposition 13. *Maximising the product-based fairness is sufficient for minimising the makespan, subjective to the load balancing problem constraints (Eq.* (3) *and* (4)*):*

$$\max F(\boldsymbol{l}) \Rightarrow \min \max(\boldsymbol{l}) \tag{28}$$

Proof. For a vector of workloads $\boldsymbol{l} = [l_1, \ldots, l_N]$ on each server $j \in [N]$, by the definition of fairness,

$$\max F(\boldsymbol{l}) = \max \frac{\prod_{j\in[N]} l_j}{\max_{k'\in[N]} l_{k'}} \tag{29}$$

WLOG, let $l_k = \max_{k'\in[N]} l_{k'}$, then,

$$\max F(\boldsymbol{l}) = \max \prod_{j\in[N], j\neq k} l_j \tag{30}$$

Similar to the proof of Proposition 12, given the stability constraint in Eq. (3) $\sum_{i=1}^{M} w_i(t) \leq \sum_{j=1}^{N} v_j$, we denote the total amount of workload in the system $C = \sum_{j=1}^{N} l_j$. Based on the constraint in Eq. (4), we have $C \geq 0$, $l_j(t) \geq 0$. By means inequality,

$$\left(\prod_{j\in[N], j\neq k} l_j\right)^{\frac{1}{N-1}} \leq \frac{\sum_{j\in[N], j\neq k} l_j}{N-1} = \frac{C - l_k}{N-1}. \tag{31}$$

with the equivalence achieved when $l_i = l_j, \forall i, j \neq k, i, j \in [N]$ holds. Therefore,

$$\max F(\boldsymbol{l}) \Rightarrow \max \frac{C - l_k}{N-1} \tag{32}$$

$$\Leftrightarrow \min l_k \tag{33}$$

$$\Leftrightarrow \min \max_{j\in[N]} l_j \tag{34}$$

The inverse may not hold since $\max \frac{C-l_k}{N-1}$ does not indicates $\max F(\boldsymbol{l})$, so maximising the linear product-based fairness is sufficient but not necessary for minimising the makespan. This finishes the proof. □

B.3 VBF for MPG

Theorem 14. *Multi-agent load balancing is MPG with the VBF $F_i(\boldsymbol{l}_i)$ as the reward r_i for each LB agent $i \in [M]$, then suppose for $\forall s \in \mathcal{S}$ at step $h \in [H]$, the potential function is time-cumulative total fairness: $\phi^{\pi_i,-\pi_i}(s) = \sum_{t=h}^{H} F^{\pi_i,-\pi_i}(\boldsymbol{l}(t))$.*

Proof.

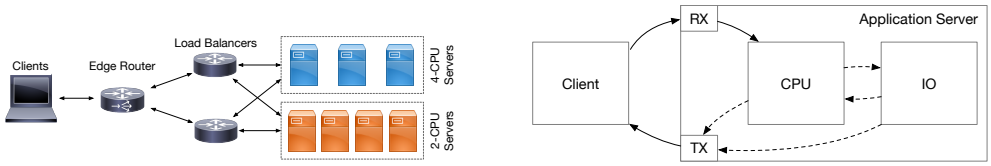
$$V_i^{\pi_i,\pi_{-i}}(s) - V_i^{\tilde{\pi}_i,\pi_{-i}}(s) = \mathbb{E}_{\pi_i,\pi_{-i}}\left[\sum_{t=h}^{H} r_{i,t}(s_t, \boldsymbol{a}_t)\middle| s_h = s\right] - \mathbb{E}_{\tilde{\pi}_i,\pi_{-i}}\left[\sum_{t=h}^{H} r_{i,t}(s_t, \tilde{a}_{i,t}, a_{-i,t})\middle| s_h = s\right] \tag{35}$$

$$= \mathbb{E}_{\pi_i,\pi_{-i}}\left[\sum_{t=h}^{H} F_i(\boldsymbol{l}_i(t))\right] - \mathbb{E}_{\tilde{\pi}_i,\pi_{-i}}\left[\sum_{t=h}^{H} F_i(\tilde{\boldsymbol{l}}_i(t))\right] \tag{36}$$

$$= \sum_{t=h}^{H}\left(F^{\pi_i,-\pi_i}(\boldsymbol{l}) - F^{\tilde{\pi}_i,-\pi_i}(\tilde{\boldsymbol{l}})\right) \quad \text{(Lemma 3)} \tag{37}$$

$$= \phi^{\pi_i,-\pi_i}(s) - \phi^{\tilde{\pi}_i,-\pi_i}(s) \tag{38}$$

Notice that s is the ground truth state of the environment, therefore involving the expected time $\boldsymbol{l}$ to finish remaining jobs. □



(a) An example of network topology with two groups of 7 servers.

(b) Illustration of the processing states of connection requests. Solid and dashed arrows represent deterministic and non-deterministic procedures respectively.

Figure 5: Simulator implementation details.

Lemma 15. *NE for MPG is ϵ-approximate NE for ϵ-approximate MPG. [46]*

Proof. We know NE (π_i^*, π_{-i}^*) for MPG,

$$V_i^{\pi_i^*,\pi_{-i}^*}(s) - V_i^{\tilde{\pi}_i,\pi_{-i}^*}(s) = \phi^{\pi_i^*,\pi_{-i}^*}(s) - \phi^{\tilde{\pi}_i,\pi_{-i}^*}(s) \geq 0 \tag{39}$$

the policies can be ϵ-approximate NE for another game with a different value function $\widehat{V}$ but the same potential function,

$$\widehat{V}_i^{\pi_i^*,\pi_{-i}^*}(s) - \widehat{V}_i^{\tilde{\pi}_i,\pi_{-i}^*}(s) \geq \epsilon, \forall i \in [N], \tilde{\pi}_i \in \Pi_i, s \in \mathcal{S} \tag{40}$$

thus,

$$\left| \left(\widehat{V}_i^{\pi_i^*,\pi_{-i}^*}(s) - \widehat{V}_i^{\tilde{\pi}_i,\pi_{-i}^*}(s) \right) - \left(\phi^{\pi^*,\pi_{-i}^*}(s) - \phi^{\tilde{\pi},\pi_{-i}^*}(s) \right) \right| \leq \epsilon \tag{41}$$

which satisfies the definition of ϵ-approximate MPG. □

C Implementation

C.1 Simulator

In order to compare the proposed RLB algorithms to the theoretically optimal solution which has perfect observation over the system – which is not achievable in real-world system, we implement an event-driven simulator to simulate the discrete process of network flow arrival and departure in a networked system. The simulator implements the network topology as in Fig. 5a, where each LB is connected to all servers.

Real-world network applications can be CPU-bound or IO-bound [47, 48]. The simulator allows configuring applications that require multi-stage processes switching between CPU/IO queues (Fig. 5b). For instance, a connection request for a 2-stage application is first processed in the CPU queue, then in the IO queue, before being sent back to the client.

Two different processing models are used for CPU and IO queues, respectively. A FIFO model is defined for CPU queues, and connections that arrive when no CPU is available will be blocked in a backlog queue until there is an available CPU. Realistic network applications feature blocked processor sharing model [47, 48], in which the instantaneous processing speed for each task $\hat{v}_j(t)$ at time t on the j-th server is:

$$\hat{v}_j(t) = \begin{cases} 1 & |w_j(t)| \leq p_j, \\ \frac{p_j}{\min(\hat{p}_j, |w_j(t)|)} & |w_j(t)| > p_j, \end{cases} \tag{42}$$

where $|w_j(t)|$ denotes the number of on-going tasks, and p_j denotes the number of processors on the j-th server. At any given moment, the maximum number of tasks that can be processed is $\hat{p}_j$. Tasks that arrive when $|w_j(t)| \geq \hat{p}_j$ will be blocked in a wait queue (similar to backlog in *e.g.* Apache) and will not be processed until there is an available slot in the CPU processing queue. However, this does not happen under the constraints in Eq. (3) as the task arrival rates are always slower than task departure rates (processing speed). The server processing speed therefore is $v_j(t) = \hat{v}_j(t)|w_j(t)|$. IO is simulated as a simple processor sharing model, in which the instantaneous processing speed is the inverse of the number of connections in the IO queue. The backlog queue length of each server is configured as 64. Connections that arrive when the backlog queues are full will be rejected, with 40s timeout. Communication latency between 2 nodes is uniformly distributed between 0.1ms and 1ms.

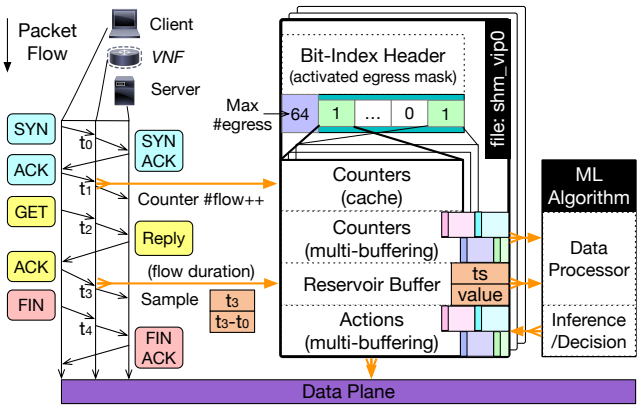


Figure 6: Feature collection mechanism: `shm` layout and data flow pipeline.

C.2 Real-World DC Testbed

C.2.1 System Platform

Application servers are virtualised on 4 UCS B200 M4 servers, each with one Intel Xeon E5-2690 v3 processor (12 physical cores and 48 logical cores), interconnected by UCS 6332 16UP fabric. Operating systems are `Ubuntu 18.04.3 LTS` (`GNU/Linux 4.15.0-128-generic x86_64`). Compilers are `gcc` version `7.5.0` (`Ubuntu 7.5.0-3ubuntu1 18.04`). Applications employed in this paper are the following: `Apache 2.4.29`, `VPP v20.05`, `MySQL 5.7.25-0ubuntu0.18.04.2`, and `MediaWiki v1.30`. The VMs are deployed on the same layer-2 link, with statically configured routing tables. Apache HTTP servers share the same VIP address on one end of GRE tunnels with the load balancer on the other end.

C.2.2 Apache HTTP Servers

The Apache servers use `mpm_prefork` module to boost performance. Each server has maximum 32 worker threads and TCP backlog is set to 128. In the Linux kernel, the `tcp_abort_on_overflow` parameter is enabled, so that a TCP RST will be triggered when the queue capacity of TCP connection backlog is exceeded, instead of silently dropping the packet and waiting for a SYN retransmit. With this configuration, the FCT measures application response delays rather than potential TCP SYN retransmit delays. Two metrics are gathered as ground truth server load state on the servers: CPU utilization and instant number of Apache busy threads. CPU utilization is calculated as the ratio of non-idle cpu time to total cpu time measured from the file `/proc/stat` and the number of Apache busy threads is assessed via Apache's *scoreboard* shared memory.

C.2.3 24-Hour Wikipedia Replay Trace

To create Wikipedia server replicas, an instance of MediaWiki[6] of version 1.30, a MySQL server and the `memcached` cache daemon are installed on each of the application server instance. *WikiLoader* tool [49] and a copy of the English version of Wikipedia database [43], are used to populate MySQL databases. The 24-hour trace is obtained from the authors of [43] and for privacy reasons, the trace does not contain any information that exposes user identities.

C.2.4 Feature Collection and Policy Update in the Data Plane

Algorithm 3 Reservoir sampling with no rejection

```
k ← reservoir buffer size
buf ← [(0,0), ..., (0,0)]                ▷ Size of k
for each observed sample v arriving at t do
    randomId ← rand()
    idx ← randomId%N                     ▷ randomly select one index
    buf[idx] ← (t, v)                    ▷ register sample in buffer
end for
```

In order to apply RL in an asynchronous close-loop load balancing framework with high scalability and low latency, communication between the load balancer data plane and the ML application is implemented via POSIX shared memory (`shm`). This mechanism allows features to be extracted from the data plane and conveyed to the RL agent – with absolutely zero control message or communication overhead, and allows data-driven decisions generated by the RL agent to be updated asynchronously on the load balancer.

The pipeline of the data flow over the lifetime of a TCP connection is depicted in Fig. 6. By statefully tracking flow states, on receipt of different networking packets, we inspect packet header and collect networking features as counters or samples. Quantitative features (task duration and task completion time) are collected as samples, using reservoir sampling (Algorithm 3). Since networking environments are dynamic, it is important to capture not only the features, but also the sequential information of the system. Reservoir sampling gathers a representative group of samples in fix-sized buffer from a stream, with $\mathcal{O}(1)$ insertion time. It captures both the sampling timestamps and exponentially-distributed numbers of samples over a time window, which help conduct sequential pattern analysis[7]. For a Poisson stream of events with rate λ, the expectation of the amount of samples that are preserved in buffer after n steps is $E = \lambda \left(\frac{k-1}{k}\right)^{\lambda n}$, where k is the size of reservoir buffer. On the other hand, counters are collected atomically and made available to the data processing agent using multi-buffering.

Cloud services have different characteristics and they are identified by virtual IPs (VIPs), which correspond to clusters of provisioned resources – *e.g.* servers, identified by a unique direct IP (DIP). In production, cloud DCs are subject to high traffic rates and their environments and topologies change dynamically. This requires to organise collected features in a generic yet scalable format, and make features available for ML algorithms without disrupting the data plane. We organise observations of each VIPs in independent POSIX shared memory (`shm`) files, to provide scalable and dynamic service management. In each `shm` file, collected features are further partitioned by egress equipments so that spatial information can be distinguished, including counters and reservoir samples. Fig. 6 exemplifies the `shm` layout and data flow.

The bit-index binary header helps efficiently identify active application servers. Each server has its own independent memory space, storing counters, reservoir samples, and data plane policies (actions) if necessary. As depicted in Fig. 6, on receipt of the first `ACK` from the client to a specific server i, VNF increments the number of flows in the counters cache of node i with $\mathcal{O}(1)$ complexity. With the same level of complexity, quatitative features (*e.g.* flow duration $t_3 - t_0$ gathered at t_3 in Fig. 6) can be stored in the reservoir buffer of node i using Algorithm 3. Gathered features (counters and samples) are made available in an organised layout and they can be quickly accessed by ML algorithms running in a different process. With the bit-index header, locating features for a given server requires $\mathcal{O}(1)$ computational complexity and $\mathcal{O}(k)$ memory complexity, where k is the reservoir buffer size. Obtained features for all active servers can then be aggregated and processed to make further inferences or data-driven decisions, which can be written back to the memory space of each server ($\mathcal{O}(1)$ computational complexity).

[6] https://www.mediawiki.org/wiki/Download

[7] Based on the characteristics of different system dynamics, *e.g.* long-term distribution shifts or short-term oscillations, tuning the reservoir sampling mechanism enables to collect different statistical representations of the states.

Operation / Complexity		Computation	Memory
Add / Remove VIP		$\mathcal{O}(1)$	$\mathcal{O}(kN + mN)$
Add server		$\mathcal{O}(1)$	$\mathcal{O}(k + m)$
Remove server		$\mathcal{O}(1)$	$\mathcal{O}(1)$
Register reservoir sample Update counter (cache)		$\mathcal{O}(1)$	$\mathcal{O}(1)$
Update counters / actions (multi-buffering)		$\mathcal{O}(1)$	$\mathcal{O}(N)$
Get the latest observation	1 node	$\mathcal{O}(m)$	$\mathcal{O}(k + m)$
	All nodes		$\mathcal{O}(kN + mN)$
Update action in the data plane	1 node	$\mathcal{O}(m)$	$\mathcal{O}(1)$
	All nodes		$\mathcal{O}(N)$

Table 7: Computation and memory complexity of different operations, where k is the size of reservoir buffer, N is the number of servers, and m is the level of multi-buffering.

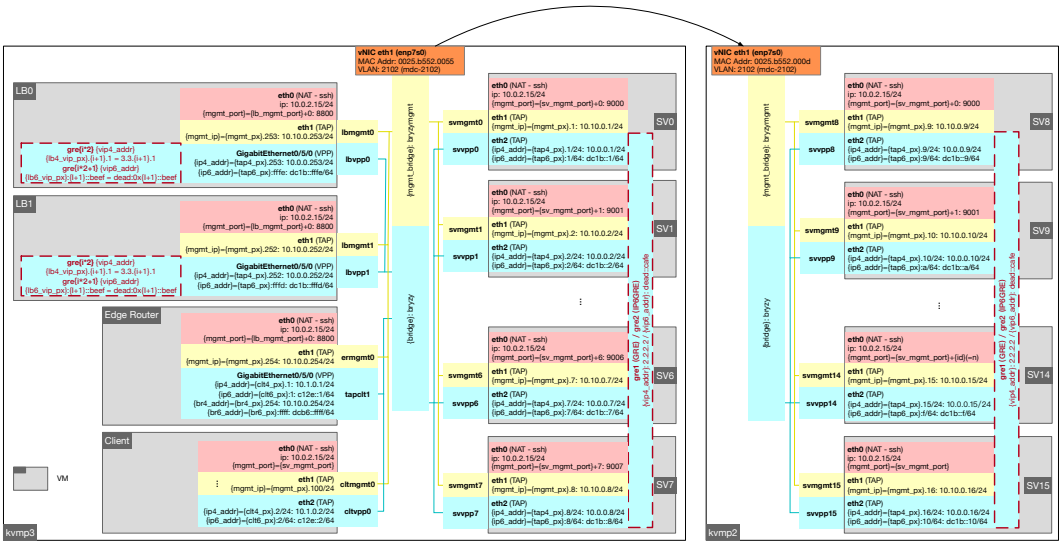


Figure 7: Network topology of the real-world DC testbed.

While quantitative features are collected using reservoir sampling, counters are incremented by the data plane in the cache, and periodically drawn from cache using m-level multi-buffering with incremental sequence ID. When copying data between cache and buffer, the sequence ID is set to 0 to avoid I/O conflicts. Pulling the counters from cache to multi-buffering requires $\mathcal{O}(1)$ computational complexity and maximal $\mathcal{O}(N)$ memory complexity. ML algorithms can pull the latest observations from the multi-buffering with no disruption in the data plane, with $\mathcal{O}(m)$ computational complexity to find the buffer with the highest sequence ID. Similarly, new network policies and data-driven decisions (*e.g.* forwarding rules) can be updated to the data plane via action multi-buffering with $\mathcal{O}(m)$ computational complexity.

To summarise, both computation and memory space complexity is presented in Table 7. The whole dataflow is asynchronous and avoid stalling in the data exchange process in both the data plane and the control plane.

C.2.5 Network Topology

For reproducibility, the network topology is depicted in Fig. 7. Two physical servers are connected via a VLAN. Each device is an instance of KVM, which is widely used for in-production vitualised Content Delivery Networks (CDNs).

Table 8: Survey on real-world testbed configurations.

Related Work	Testbed Scale	Note
6LB [4]	2 LB + 28 servers (2-CPU each)	Our paper uses the same network trace as input traffic.
Ananta [50]	14 LBs for 12 VIPs	The exact number of servers per VIP and the in-production traffic is not documented in the paper.
Beamer [51]	2 LB + 8 servers (small) 4 LB + 10 servers (large)	Large scale experiments are conducted with 700 active HTTP connections max.
Duet [52]	3 software LB + 3 hardware LB + 34 servers	Synthetic traffic is applied so that the server cluster behind VIP processes 60k (identical) packets per second.
SilkRoad [14]	1 hardware LB or 3 software LB per VIP	Real-world PoP traffic is applied, where one server cluster behind VIP processes on average 309.84 active connections per second.
Cheetah [20]	2 LB + 24 servers	A Python generator creates 1500-2500 synthetic requests/s.
Our Paper	2 LB + 8 servers (simulation) 2 LB + 7 servers (small) 6 LB + 20 servers (large)	Our paper is representative of real-world circumstances not only in terms of number of agents and servers, but also in terms of traffic rates (> 1150.76 concurrent connections).

C.2.6 Realistic Testbed

Modern data center may comprise thousands of servers and hundreds of LBs. However, each independent service is exposed in a modular way at one or several virtual IP (VIP) addresses to receive requests, running over a cluster of servers. Each server in the cluster can be identified by a unique direct IP (DIP). Traffic and queries from the clients destined to a VIP are load balanced among the DIPs of the service. The development of virtualization, where computers are emulated and/or sharing an isolated portion of the hardware by way of Virtual Machines (VMs), or run as isolated entities (containers) within the same operating system kernel, has accelerated the commoditization of compute resources. Therefore, the gigantic in-production data center network are typically partitioned into small pods, where different services (VIPs) are hosted. To justify the setups of our experiment satisfy the "real-world" requirement, we present a brief survey of real-world DC setup based on a set of state-of-the-art load balancing research papers, which are summarized below (Table 8).

Using 2 physical servers (48 CPUs each), we have made our best effort to find a configuration that allows us to conduct experiments similar to real-world setups. Based on the survey above, we believe that the experiments conducted in this paper have reasonable scale – not only in terms of number of agents ($2/6$ LBs) and servers ($7/20$ servers), but also in terms of traffic rates – more than 2k queries per second per VIP and more than 1150.76 concurrent connections in large scale experiments —- and are representative of real-world circumstances.

C.3 Benchmark Load Balancing Methods

To compare load balancing performance, 4 state-of-the-art workload distribution algorithms are implemented. Equal-cost multi-path (ECMP) randomly assigns servers to tasks with a server assignment function $\mathbb{P}(j) = \frac{1}{n}$, where $\mathbb{P}(j)$ denotes the probability of assigning the j-th server [13]. Weighted-cost multi-path (WCMP) assigns servers based on their weights derived, and has an assignment function as $\mathbb{P}(j) = \frac{v_j}{\sum v_j}$ [3]. Local shortest queue (LSQ) assigns the server with the shortest queue, *i.e.* $\arg\min_{j\in[n]} |w^j(t)|$ [19]. Shortest expected delay (SED) assigns the server the shortest queue normalized by the number of processors, *i.e.* $\arg\min_{j\in[n]} \frac{|w^j(t)|+1}{v_j}$ [2], and is expected to have the best performance among conventional heuristics. In the simulator, an *Oracle* LB algorithm is implemented, which distributes connections to the server which is expected to finish all its job with the lowest delay (including the new connection). The Oracle LB is aware of the remaining time of each connection, which is otherwise not observable for network LBs in real-world setups. When receiving a new connection, the Oracle LB algorithm calculates the remaining time to process on each server (assuming the newly received connection is assigned on the server as well) and assigns the server with the lowest remaining time to process to the new-coming connection, to make sure that the makespan is always minimised with the global observation, which is not possible to be achieved in real-world system. The load balancing decisions for the Oracle algorithm are also made immediately for the Oracle LB algorithm.

Table 9: Hyperparameters in MARL-based LB.

	Hyperparameter	Simulation	Experiments	
		Moderate-Scale	Moderate-Scale	Large-Scale
Distributed LB	Learning rate	3×10^{-4}	1×10^{-3}	1×10^{-3}
	Batch size	25	25	12
	Hidden dimension	64	64	128
	Hypernet dimension	32	32	64
	Replay buffer size	3000	3000	3000
	Episodes	500	120	200
	Updates per episode	10	10	10
	Step interval	0.5s	0.25s	0.25s
	Target entropy	$-\|\mathcal{A}\|$	$-\|\mathcal{A}\|$	$-\|\mathcal{A}\|$
LB System	TCT Distribution	Exponential	Real-world trace	Real-world trace
	Average TCT	1s	200ms	200ms
	Average bytes per task	-	12KiB	12KiB
	Traffic rate	20.28tasks/s	$[650, 800]$tasks/s	2000tasks/s
	Number of LB agents	2	2	6
	Total number of servers	8	7	20
	Server group 2	4 (1-CPU)	3 (2-CPU)	10 (2-CPU)
	Server group 1	4 (2-CPU)	4 (4-CPU)	10 (4-CPU)
	Episode duration	60s	60s	60s

Table 10: Four configurations with different application types.

Application Type	Pure CPU	CPU Intensive	Balanced	IO Intensive
Avg. CPU Time (s)	1.	0.75	0.5	0.25
Avg. IO Time (s)	0.	0.25	0.5	0.75

D Hyperparameters

MARL-based load balancing methods are trained in both simulator, and moderate- and large-scale testbed setups for various amount of episodes. At the end of each episode, the RL models are trained and updated for 10 iterations. Given the total provisioned computational resource, the traffic rates of network traces for training are carefully selected so that the RL models can learn from sensitive cases where workloads should be carefully placed to avoid overloaded less powerful servers. The set of hyper-parameters are listed in Table 9.

E Results

E.1 Inaccurate Server Weights

In real-world systems, not only error-prone configurations, but also different application profiles can lead to inaccurate server weight assignments. Using a similar setup where 2 cluster of 4 servers have the same IO processing speed but 2x different CPU processing speeds, different application profiles are compared to derive the actual server processing capacity differences. A 3-stage application whose queries follow CPU-IO-CPU processing stages is compared with a pure CPU application. Both CPU and IO processing time follow exponential distributions and the aggregated average FCT is 1s. The four different types of network applications are configured as in Table 10. As depicted in Fig. 2a, with different provisioned resource ratios for CPU (`2x`) and IO (`1x`) queues, to guarantee the optimal workload distribution fairness and make each server have the minised maximal remaining time to finish among all servers at all time, the weights to be assigned for servers with different provisioned capacities are stochastic and depend on different application profiles. Therefore, it is a sub-optimal solution for existing load balancing algorithms to statically configure server weights based on computational resources.

The setup in the paper for Table 3 is the following. There are 2 LB agents and 8 servers. 4 servers have 1 CPU worker-thread each while the other 4 servers have 2 CPU worker-threads each, to simulate

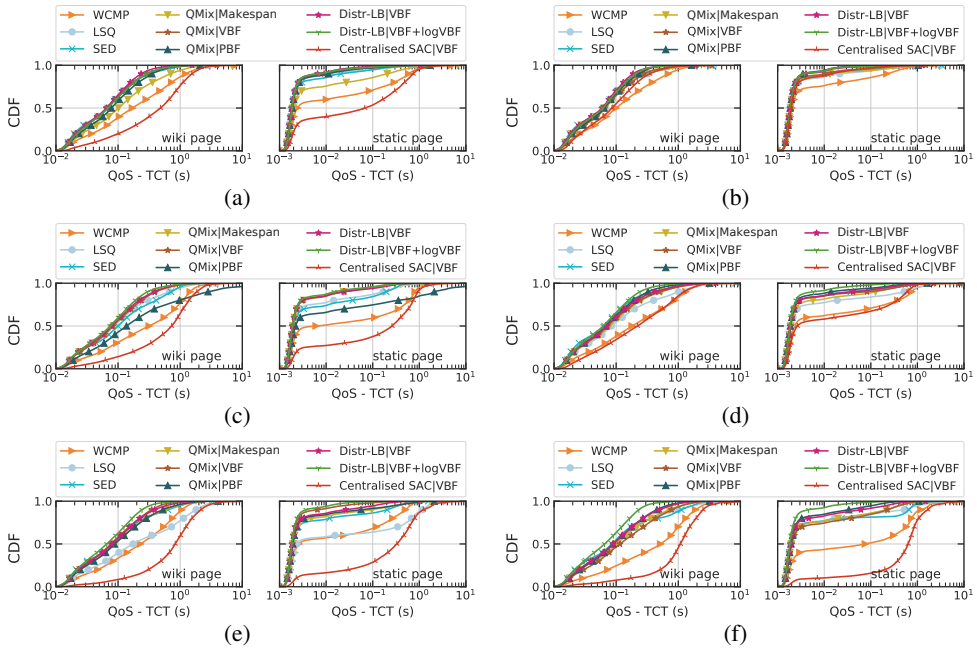


Figure 8: Experimental results with real-world network traces from different period of time during a day, which demonstrates the effectiveness of the proposed distributed RL framework with VBF as rewards.

the different server processing capacities. Three types of applications are compared. 100%-CPU application is a single stage application, whose expected time to process is 1s in the CPU queue and 0s in the IO queue. 75%-CPU$+25\%$-IO application is a two stage application, whose expected time to process is 0.75s in the CPU queue and 0.25s in the IO queue, simulating the CPU-intensive applications. 50%-CPU$+50\%$-IO application is a two stage application, whose expected time to process is 0.5s in both the CPU and IO queue. The actual time to process for each task follows an exponential distribution. The traffic rate is normalised to consume on average 84.5% resources.

E.2 Ablation Results

Besides the experiments conducted in the paper, we further study the following aspects of the application of MARL on real-world network load balancing problems.

E.2.1 Reward Engineering

To verify the effectiveness of the proposed potential function VBF, we compare it with a set of different reward functions, including makespan (MS), PBF, and coefficient of variation (CV). During our study based on real-world testbed, we found that, when using VBF as the reward, the convergence is fast at the beginning of the training process and the sample variance of average flow duration (as an estimation of the queuing and processing delay) on each server becomes close to zero. However, it does not necessarily mean that the load balancing policy is optimal and the NE is achieved. To capture the small variance which is close-to-zero, we also calculate the logarithm of VBF ($\log$VBF) as reward. And the combination of VBF + $\log$VBF is an empirical design aiming at faster convergence towards the NE policy. The complete comparison results are shown in Table 11 (average QoS) and in Table 12 (99th percentile QoS), where the proposed distributed MARL framework achieves the best performance for most cases. To provide a complete view of all comparison results besides the one shown in Fig. 4b, we show the CDF of task completion time under all test cases in Fig. 8 Accompanying the evaluation results of average QoS in large-scale testbed in Table 4, we also show in Table 13 the 99th percentile QoS in large-scale testbed.

Table 11: Complete results of *average* QoS (s) for comparison in moderate-scale real-world network setup (DC network and traffic).

Method		Period I (796.315 queries/s)		Period II (687.447 queries/s)		Period III (784.522 queries/s)		Period IV (784.522 queries/s)	
		Wiki	Static	Wiki	Static	Wiki	Static	Wiki	Static
WCMP		0.435 ± 0.083	0.171 ± 0.055	0.254 ± 0.087	0.073 ± 0.056	0.412 ± 0.101	0.134 ± 0.059	0.834 ± 0.323	0.492 ± 0.276
LSQ		0.141 ± 0.073	0.023 ± 0.030	0.143 ± 0.040	0.023 ± 0.011	0.620 ± 0.442	0.339 ± 0.316	0.357 ± 0.373	0.173 ± 0.299
SED		0.137 ± 0.076	0.020 ± 0.023	0.131 ± 0.067	0.027 ± 0.035	0.215 ± 0.210	0.051 ± 0.081	0.346 ± 0.496	0.169 ± 0.330
RLB-SAC [40]	Jain	0.137 ± 0.020	0.009 ± 0.006	0.125 ± 0.035	0.012 ± 0.008	0.193 ± 0.073	0.026 ± 0.022	0.204 ± 0.084	0.039 ± 0.047
	G	0.140 ± 0.053	0.015 ± 0.019	0.103 ± 0.022	0.010 ± 0.007	0.149 ± 0.049	0.015 ± 0.011	0.155 ± 0.052	0.011 ± 0.011
QMix-LB	MS	0.258 ± 0.174	0.071 ± 0.087	0.142 ± 0.073	0.030 ± 0.034	0.217 ± 0.157	0.048 ± 0.069	0.263 ± 0.202	0.073 ± 0.092
	logMS	0.167 ± 0.031	0.009 ± 0.004	0.132 ± 0.034	0.011 ± 0.008	0.844 ± 1.376	0.635 ± 1.249	0.278 ± 0.130	0.041 ± 0.038
	VBF	0.128 ± 0.052	0.014 ± 0.017	0.132 ± 0.075	0.016 ± 0.025	0.141 ± 0.025	0.008 ± 0.004	0.286 ± 0.162	0.068 ± 0.066
	logVBF	$\mathbf{0.106 \pm 0.011}$	0.007 ± 0.001	0.109 ± 0.032	0.011 ± 0.009	0.171 ± 0.043	0.022 ± 0.013	0.223 ± 0.045	0.026 ± 0.017
	VBF+logVBF	0.112 ± 0.005	$\mathbf{0.005 \pm 0.002}$	0.101 ± 0.010	0.005 ± 0.001	0.187 ± 0.090	0.024 ± 0.029	0.201 ± 0.080	0.021 ± 0.020
	PBF	0.142 ± 0.035	0.012 ± 0.006	0.099 ± 0.011	$\mathbf{0.004 \pm 0.001}$	0.211 ± 0.153	0.047 ± 0.078	0.181 ± 0.042	0.018 ± 0.009
	CV	0.407 ± 0.505	0.201 ± 0.340	0.113 ± 0.036	0.009 ± 0.008	0.203 ± 0.089	0.039 ± 0.037	0.219 ± 0.072	0.031 ± 0.017
Centr-LB	VBF	0.690 ± 0.211	0.284 ± 0.181	0.152 ± 0.041	0.016 ± 0.011	1.068 ± 0.386	0.570 ± 0.378	1.378 ± 0.377	0.867 ± 0.350
	logVBF	0.676 ± 0.231	0.265 ± 0.151	0.160 ± 0.023	0.013 ± 0.005	0.938 ± 0.200	0.446 ± 0.179	0.972 ± 0.288	0.495 ± 0.268
	VBF+logVBF	0.520 ± 0.034	0.167 ± 0.017	0.192 ± 0.040	0.019 ± 0.014	0.759 ± 0.254	0.306 ± 0.222	1.013 ± 0.168	0.520 ± 0.167
Distr-LB (this paper)	VBF	$\mathbf{0.106 \pm 0.013}$	0.007 ± 0.002	$\mathbf{0.090 \pm 0.016}$	0.007 ± 0.005	0.159 ± 0.054	0.017 ± 0.009	0.196 ± 0.091	0.032 ± 0.033
	logVBF	0.139 ± 0.021	0.011 ± 0.004	0.129 ± 0.032	0.012 ± 0.011	0.250 ± 0.156	0.057 ± 0.077	0.226 ± 0.059	0.038 ± 0.019
	VBF+logVBF	0.126 ± 0.038	0.009 ± 0.006	0.094 ± 0.023	0.006 ± 0.006	$\mathbf{0.108 \pm 0.022}$	$\mathbf{0.004 \pm 0.001}$	$\mathbf{0.104 \pm 0.013}$	$\mathbf{0.006 \pm 0.003}$
	CV	0.150 ± 0.040	0.011 ± 0.009	0.149 ± 0.060	0.026 ± 0.025	0.301 ± 0.146	0.066 ± 0.072	0.267 ± 0.156	0.051 ± 0.052

Table 12: Complete results of *99th percentile* QoS (s) for comparison in moderate-scale real-world network setup (DC network and traffic).

Method		Period I (796.315 queries/s)		Period II (687.447 queries/s)		Period III (784.522 queries/s)		Period IV (784.522 queries/s)	
		Wiki	Static	Wiki	Static	Wiki	Static	Wiki	Static
WCMP		5.801 ± 4.519	4.462 ± 3.867	4.019 ± 3.601	3.192 ± 3.543	3.239 ± 2.721	2.305 ± 2.700	8.066 ± 7.025	6.733 ± 5.329
LSQ		0.722 ± 0.487	0.195 ± 0.314	0.814 ± 0.478	0.288 ± 0.259	1.846 ± 1.915	1.168 ± 1.575	1.257 ± 1.921	0.831 ± 2.002
SED		0.706 ± 0.399	0.208 ± 0.246	0.697 ± 0.460	0.217 ± 0.291	0.726 ± 0.554	0.203 ± 0.261	0.909 ± 1.180	0.450 ± 1.112
RLB-SAC [40]	Jain	0.858 ± 0.240	0.159 ± 0.125	0.830 ± 0.358	0.227 ± 0.186	1.227 ± 0.489	0.354 ± 0.246	1.283 ± 0.594	0.408 ± 0.374
	G	0.945 ± 0.495	0.185 ± 0.214	0.682 ± 0.255	0.177 ± 0.162	1.003 ± 0.459	0.225 ± 0.176	0.973 ± 0.389	0.166 ± 0.156
QMix-LB	MS	1.469 ± 0.789	0.584 ± 0.547	1.095 ± 0.694	0.444 ± 0.423	1.182 ± 0.801	0.420 ± 0.483	1.447 ± 0.885	0.751 ± 0.772
	logMS	0.985 ± 0.264	0.117 ± 0.043	0.909 ± 0.388	0.172 ± 0.142	7.043 ± 12.237	6.427 ± 12.479	1.326 ± 0.584	0.371 ± 0.305
	VBF	0.732 ± 0.395	0.159 ± 0.239	0.665 ± 0.550	0.157 ± 0.278	0.744 ± 0.278	0.123 ± 0.093	1.028 ± 0.694	0.279 ± 0.365
	logVBF	0.682 ± 0.100	0.124 ± 0.019	0.772 ± 0.313	0.205 ± 0.159	1.174 ± 0.323	0.382 ± 0.183	1.426 ± 0.323	0.327 ± 0.153
	VBF+logVBF	0.664 ± 0.057	$\mathbf{0.087 \pm 0.056}$	0.611 ± 0.097	0.055 ± 0.027	1.171 ± 0.568	0.302 ± 0.293	1.206 ± 0.501	0.278 ± 0.239
	PBF	0.661 ± 0.193	$\mathbf{0.087 \pm 0.099}$	0.505 ± 0.119	0.048 ± 0.029	0.768 ± 0.728	0.205 ± 0.465	0.726 ± 0.433	0.128 ± 0.136
	CV	1.928 ± 2.228	1.281 ± 2.095	0.708 ± 0.405	0.131 ± 0.130	1.331 ± 0.593	0.481 ± 0.297	1.344 ± 0.329	0.451 ± 0.218
Centr-LB	VBF	3.101 ± 1.582	1.985 ± 1.790	0.903 ± 0.350	0.328 ± 0.353	4.409 ± 2.693	3.629 ± 3.219	6.649 ± 4.562	6.120 ± 4.721
	logVBF	2.715 ± 0.444	1.718 ± 0.547	1.016 ± 0.229	0.264 ± 0.092	3.247 ± 0.725	2.136 ± 0.832	4.286 ± 2.091	3.459 ± 2.323
	VBF+logVBF	2.459 ± 0.101	1.309 ± 0.063	1.243 ± 0.358	0.285 ± 0.189	2.796 ± 0.900	1.702 ± 1.287	3.466 ± 0.820	2.628 ± 1.142
Distr-LB (this paper)	VBF	$\mathbf{0.651 \pm 0.151}$	0.119 ± 0.072	0.571 ± 0.237	0.133 ± 0.136	1.039 ± 0.302	0.298 ± 0.125	1.187 ± 0.594	0.355 ± 0.318
	logVBF	0.923 ± 0.162	0.193 ± 0.086	0.933 ± 0.415	0.243 ± 0.302	1.491 ± 0.764	0.579 ± 0.531	1.481 ± 0.473	0.558 ± 0.286
	VBF+logVBF	0.745 ± 0.316	0.185 ± 0.152	$\mathbf{0.385 \pm 0.094}$	$\mathbf{0.023 \pm 0.003}$	$\mathbf{0.595 \pm 0.199}$	$\mathbf{0.051 \pm 0.030}$	$\mathbf{0.563 \pm 0.180}$	$\mathbf{0.100 \pm 0.073}$
	CV	0.865 ± 0.261	0.147 ± 0.121	1.109 ± 0.668	0.433 ± 0.431	1.730 ± 0.468	0.612 ± 0.420	1.383 ± 0.666	0.446 ± 0.345

E.2.2 Communication Overhead of CTDE and Centralised RL

This section studies the communication overhead of CTDE RL scheme and analyses its impact on real-world distributed systems.

First, we discuss the communication overhead in data center networks in two-folds: throughput and latency.

1. **Thoughput:** Active signaling (*e.g.* periodically probing, or sharing messages) is an intrinsic way to observe and measure system states so that informed decisions can be made to improve performance [53, 54]. Higher communication frequency gives more relevant and timely observations yet there is a trade-off between communication frequency and additionally consumed bandwidth. Especially, in large distributed systems like data center networks, services are organized by multiple server clusters scattered all over the physical data center network in the era of cloud computing. Thus, management traffic among different nodes can cascade and plunder the bandwidth for data transmission in high-tier links. To demonstrate the trade-off between measurement quality and throughput overhead, we have conducted experiments to evaluate (i) the relevance of collected server utilization information to the actual server utilization information with root mean square error (RMSE) and Spearman's Correlation in our testbed on physical servers. When a controller VM periodically probes a server cluster via TCP sockets[8], as depicted in Fig. 9, the visibility over system states (relevance between measurements and ground truth) correlates with the probing frequency. Additional management traffic within a single service cluster —- behind one virtual IP (VIP) —- can exceed the 90th percentile of per-destination-rack flow rate (100kbps as depicted in Figure 8a in [45]) in Facebook data center in production.

As depicted in Fig. 10a, CTDE RL scheme requires agents to communicate and share their trajectories, which include the observed states and actions. This leads to linearly increasing replay buffer size with

[8]In the 69-byte control packet emitted by the server, the 24-byte payload consists of the server ID, CPU and memory usage, and the number of busy application threads.

Table 13: Comparison of 99th percentile QoS (s) in large-scale real-world network setup (DC network and traffic).

Method		Period I (2022.855 queries/s)		Period II (2071.129 queries/s)	
		Wiki	Static	Wiki	Static
WCMP		3.014 ± 0.612	2.152 ± 0.907	4.290 ± 3.593	3.300 ± 3.308
LSQ		1.863 ± 0.888	0.843 ± 0.773	1.243 ± 1.389	0.675 ± 1.223
SED		0.891 ± 0.475	0.208 ± 0.251	1.074 ± 0.751	0.592 ± 0.650
RLB-SAC-G [40]		1.064 ± 0.283	0.210 ± 0.132	0.739 ± 0.317	0.186 ± 0.214
QMix-LB	VBF	1.104 ± 0.481	0.241 ± 0.264	1.223 ± 1.169	0.634 ± 0.983
	PBF	1.201 ± 0.321	0.196 ± 0.112	0.583 ± 0.103	0.071 ± 0.050
Distr-LB	VBF	1.350 ± 0.311	0.263 ± 0.139	1.180 ± 0.702	0.448 ± 0.371
(this paper)	VBF+logVBF	$\mathbf{0.890 \pm 0.250}$	$\mathbf{0.103 \pm 0.064}$	$\mathbf{0.531 \pm 0.149}$	$\mathbf{0.057 \pm 0.039}$

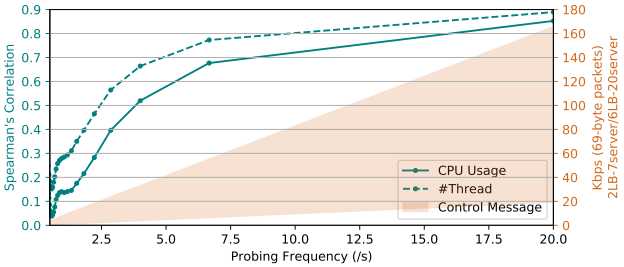


Figure 9: Correlation (Spearman) increases when the probing frequency grows, yet, so do additional control messages.

the growth of number of episodes. The replay buffer size also grows with the number of agents which makes CTDE RL scheme not a scalable mechanism. Transmitting and synchronising replay buffer among agents incur additional communication overhead in the networking system, reducing the throughput for data transmission channel – which can break full-bisection bandwidth (an important throughput related performance metric) in data center networks [55] – thus decreasing the QoS.

2. **Latency:** Using the same network topology as the moderate-scale real-world testbed, when a single controller VM periodically transmit different amount of bytes via TCP sockets towards the agents, the latency overhead increases with the number of servers, which diminishes the QoS, as depicted in Fig. 10b. It is measured for per-packet round trip time (RTT) between two directly connected network nodes. While normal RTT is $0.099ms \pm 0.014ms$ in such setup, with additional communication overhead, RTT can grow more than 10x. This is not considered as low additional latency, especially not in high performance networking systems. In elastic and cloud computing context and real-world setups, load balancers can be deployed in different racks [56]. There can be multiple hops between two nodes and one connection consists of tens of hundreds of packets, which can lead to cascaded high latency.

Based on the analysis of Fig. 9, we can see that delayed measurement and communication can cause degraded system state observation. To further demonstrate the performance of the passive feature collection mechanism which incurs absolutely zero communication overhead, an additional experiment is conducted to compare the feature collection latency. The latency overhead of passive feature collection process in our paper using POSIX shared memory is compared with different active probing techniques. The idle communication latency is compared using both KVM and Docker containers between two hosts either deployed on the same machine (local) or on two neighbor machines

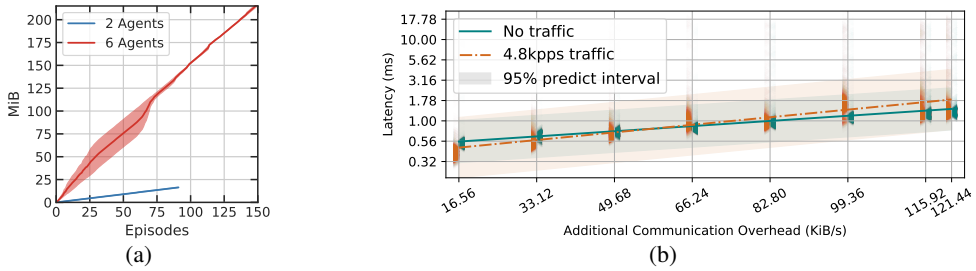


Figure 10: Communication overhead for CTDE (a) grows linearly during training and (b) can have negative effects on the packet transmission latency of the whole networking system.

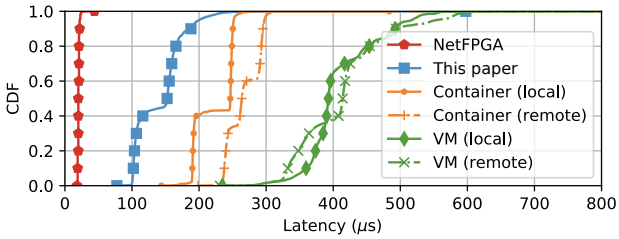


Figure 11: Feature collection latency comparison against active probing techniques.

Table 14: Comparison of average job completion time (s) of static pages under different traffic rates using large-scale real-world setup.

Method		Traffic Rate (queries/s)								
		731.534	1097.3	1463.067	1828.834	2194.601	2377.484	2560.368	2743.251	2926.135
LSQ		0.048 ±0.002	0.055 ±0.003	0.059 ±0.003	0.069 ±0.008	0.131 ±0.070	0.643 ±0.325	1.910 ±0.269	2.873 ±0.215	3.545 ±0.146
SED		0.054 ±0.001	0.061 ±0.004	0.068 ±0.004	0.080 ±0.004	0.117 ±0.025	0.660 ±0.396	1.718 ±0.366	2.767 ±0.207	3.482 ±0.189
Distr-LB (this paper)	VBF	0.047 ±0.001	0.054 ±0.003	0.059 ±0.005	**0.066 ±0.007**	0.105 ±0.035	**0.266 ±0.139**	1.465 ±0.115	2.047 ±0.145	2.704 ±0.108
	VBF+logVBF	0.047 ±0.001	0.054 ±0.004	0.059 ±0.004	0.069 ±0.008	**0.084 ±0.009**	0.413 ±0.249	**1.183 ±0.063**	**1.838 ±0.083**	**2.513 ±0.105**

(remote). To compare with the shortest latency possible of an hardware-based SDN controller directly connected to the agent, a loopback test is conducted using a NetFPGA [57] connected to the machine via both Ethernet and PCIe. We parse features stored in the local shared memory with a simple Python script without generating control messages. As depicted in Fig. 11, its median processing latency outperforms typical VM- and container-based VNF probing mechanisms [58–61] by more than $94.18\mu s$.

To evaluate the performance of the proposed algorithm within the operational range of traffic rates, we conducted the scaling experiment using 6 LB agents and 20 servers in the real-world testbed with traffic rates that range from low to high. As shown in Table 14 and 15, similar to the QoS (99-th percentile of the task completion time) evaluation in Table 5 and 6. Low traffic rates do not saturate server processing capacities and the servers are not stressed. Therefore, all servers are able to handle all the requests without accumulating jobs in the queue regardless of the differences of their processing capacities. However, under heavy traffic rates, LSQ still distribute workloads so as to maintain the same queue lengths on servers with different processing speeds, which leads to degraded average task completion time. SED assigns more jobs in proportional to the number of CPUs deployed for each server, achieving slightly better performance than LSQ in terms of the average task completion time. The proposed Distr-LB outperforms both LSQ and SED especially under heavy traffic rates, thus when servers undergo heavy resource utilisation. Since the server processing speed for different applications is not necessarily proportional to the number of CPU–as we have discussed over Fig. 2a in Sec. 2, Distr-LB is able to learn the appropriate ratio of workload distribution for servers with different capacities.

Table 15: Comparison of average job completion time (s) of static pages under different traffic rates using large-scale real-world setup.

Method		Traffic Rate (queries/s)								
		731.534	1097.3	1463.067	1828.834	2194.601	2377.484	2560.368	2743.251	2926.135
LSQ		0.004 ±0.001	0.004 ±0.000	0.003 ±0.000	0.004 ±0.000	0.018 ±0.023	0.252 ±0.234	1.455 ±0.258	2.426 ±0.207	3.080 ±0.136
SED		0.003 ±0.000	0.004 ±0.001	0.004 ±0.000	0.004 ±0.000	0.006 ±0.003	0.284 ±0.308	1.283 ±0.374	2.322 ±0.226	3.041 ±0.188
Distr-LB (this paper)	VBF	0.004 ±0.000	0.004 ±0.000	0.004 ±0.000	0.004 ±0.000	**0.005 ±0.001**	**0.055 ±0.070**	1.039 ±0.144	1.617 ±0.135	2.277 ±0.096
	VBF+logVBF	0.004 ±0.000	0.004 ±0.000	0.004 ±0.000	0.004 ±0.000	0.006 ±0.004	0.116 ±0.114	**0.750 ±0.063**	**1.413 ±0.083**	**2.076 ±0.096**

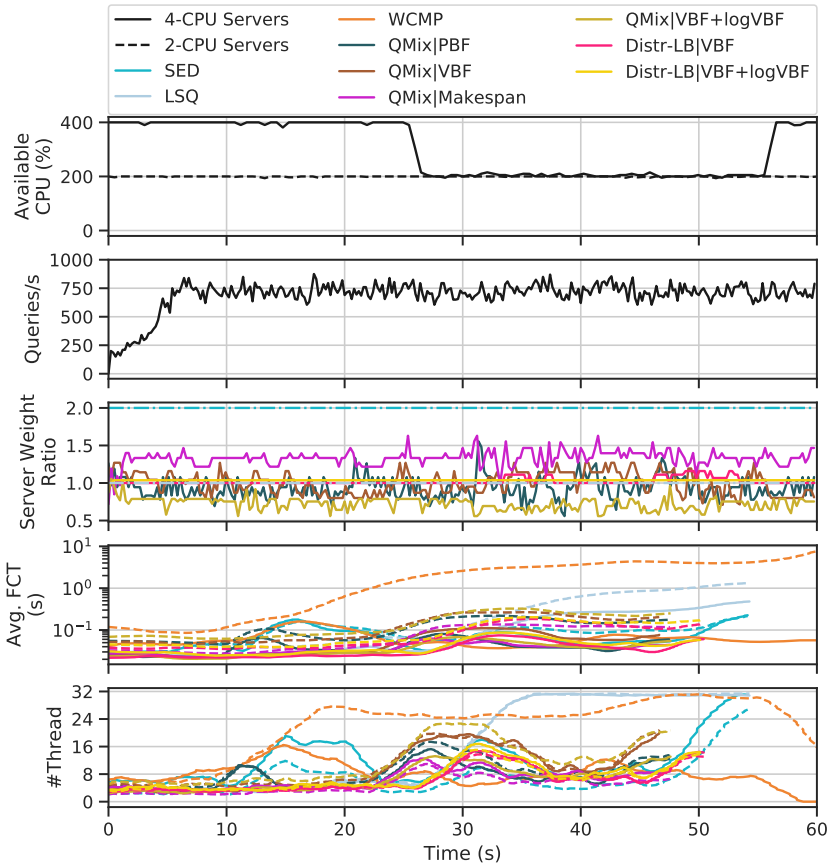
Table 16: Comparison of QoS (mean, 95th-percentile, and 99th-percentile task completion time in s) when server processing capacity changes over time.

		Wiki			Static		
		Mean	$95th$-percentile	$99th$-percentile	Mean	$95th$-percentile	$99th$-percentile
WCMP		1.792 ± 0.393	7.534 ± 1.817	2.366 ± 1.685	1.512 ± 0.385	6.571 ± 1.996	1.084 ± 1.842
LSQ		0.453 ± 0.178	1.958 ± 0.827	3.482 ± 1.257	0.202 ± 0.130	0.975 ± 0.617	1.801 ± 1.064
SED		0.340 ± 0.268	1.225 ± 0.812	30.600 ± 6.718	0.130 ± 0.206	0.519 ± 0.571	29.893 ± 7.042
QMix-LB	MS	0.373 ± 0.177	1.621 ± 0.830	4.046 ± 6.632	0.144 ± 0.112	0.663 ± 0.523	2.655 ± 6.899
	PBF	0.368 ± 0.375	1.529 ± 1.581	2.436 ± 1.468	0.159 ± 0.338	0.733 ± 1.437	0.974 ± 1.204
	VBF	0.282 ± 0.166	1.186 ± 0.799	3.187 ± 1.479	0.081 ± 0.104	0.395 ± 0.518	1.654 ± 1.181
	VBF+logVBF	0.533 ± 0.179	2.525 ± 0.913	4.864 ± 1.635	0.266 ± 0.129	1.409 ± 0.680	3.374 ± 1.626
Distr-LB (this paper)	VBF	0.262 ± 0.100	1.086 ± 0.454	2.190 ± 0.792	0.057 ± 0.044	0.305 ± 0.234	0.683 ± 0.510
	VBF+logVBF	$\mathbf{0.221 \pm 0.112}$	$\mathbf{0.895 \pm 0.530}$	$\mathbf{1.903 \pm 0.976}$	$\mathbf{0.039 \pm 0.057}$	$\mathbf{0.197 \pm 0.284}$	$\mathbf{0.480 \pm 0.650}$

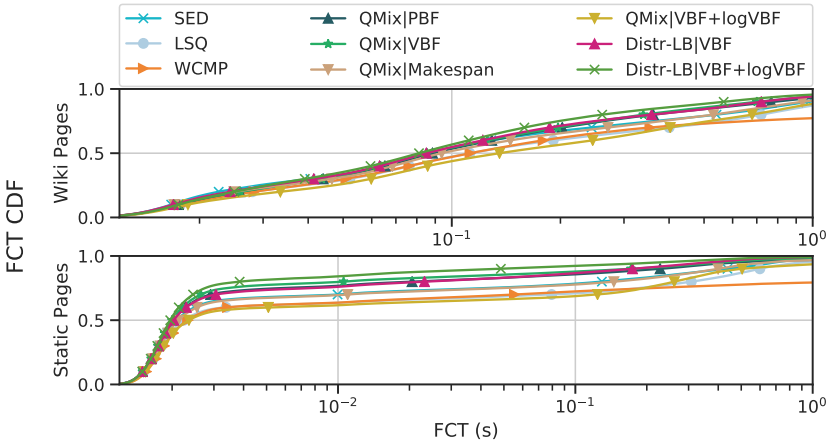
E.2.3 MARL Robustness

With the rise of elastic and server-less computing, where tenants in data center can share physical resources (*e.g.* CPU, disk, memory), servers can have different processing capacities, which may also change over time dynamically —- because of *e.g.* updated server configuration (upgrading an Amazon EC2 `a1.xlarge` instance to `a1.4xlarge`) or resource contention (co-located workloads) [62]. According to [14], there are 32% of server clusters in data center that update more than 10 times per minute based on the measurements collected over 432 minutes up time in a month. 3% of clusters have more than 50 updates perf minute. Therefore, dynamic changes prevail in real-world data center networks.

Therefore, this section studies the robustness of the proposed distributed RL-based LB framework to react to dynamic changes in server processing speeds, *e.g.* when server VMs are migrated to a new physical architecture. Using the same moderate-scale real-world testbed with 2 LB agents, additional CPU-bound workloads are applied on the 4-CPU server group starting from 25s. As depicted in Fig. 12a, under heavy Wikipedia traffic, MARL-based LB agents adapt server weights over time and achieves better performance than heuristic LB algorithms – finishing the same amount of workloads faster, maintaining lower amount of acive number of threads, even when server processing capacity is reduced. As depicted in Fig. 12b, over multiple runs (10 runs for each LB algorithm), RL-based LB algorithms effectively achieves lower task completion time in dynamic environments. They help avoid human intervention and make the LB agents autonomously adapt to the changes in the system. Table 16 lists the performance of all LB algortihms in terms of the QoS (measured as the average and 95th-percentile task completion time).



(a) Additional workloads are applied on servers with 4 CPUs at around 25s.



(b) FCT CDF comparisons for two types of tasks.

Figure 12: Load balancing performance comparison in dynamic environments.