

Deep Reinforcement Learning for Dynamic Load Balancing in Distributed Computing

Hariharan P^{1*}, Chandrasekar N² and Padmanabhan P³

¹Department of Computational Intelligence, Pamba Institute of Data-Driven Systems, Malaysia Email:

praveen.hariharan@pamba.edu

ORCID: <https://orcid.org/3934-5103-1498-1469>

²Department of Artificial Intelligence and Data Science, Periyar Institute of Autonomous Systems Research, Vietnam

Email: naveen.chandrasekar@periyar.edu

ORCID: <https://orcid.org/4518-3440-3131-6563>

³Department of Information Technology, Pamba Institute of Data-Driven Systems, Bhutan Email:

priyanka.padmanabhan@pamba.edu

ORCID: <https://orcid.org/5349-2065-1321-0410>

ABSTRACT

Distributed systems have long balanced load with simple, stateless heuristics -- round robin, least connections, power-of-two-choices -- that react to a snapshot of current server load rather than learning how a system actually behaves over time under its own particular traffic. Deep reinforcement learning proposes a different approach: an agent that observes system state, routes work accordingly, and improves its policy from the rewards that routing decision produces. This article synthesizes a decade of applied deep reinforcement learning research on dynamic load balancing across four layers of the distributed-computing stack -- network-level connection routing, cluster job scheduling, container-orchestration autoscaling, and wide-area traffic engineering -- tracing the underlying algorithmic toolkit from tabular Q-learning through deep Q-networks, double and dueling refinements, and policy-gradient and actor-critic methods including DDPG, A3C, and PPO. We examine how each layer formulates load balancing as a Markov decision process, review reported results including a multi-agent QMIX load balancer's 1.44x faster tail latency over a classical baseline and a graph-neural deep-RL traffic-engineering system's 197-625x speedup over competing optimizers, and close with an honest accounting of the field's open problems: thin evaluation realism, an underspecified safety story for policies that keep learning in production, multi-agent credit-assignment difficulty, and inconsistent benchmarking practice. Read together, the evidence supports deep reinforcement learning as a demonstrably productive, if still maturing, architectural pattern for dynamic load balancing in distributed computing.

Keywords: deep reinforcement learning; dynamic load balancing; distributed computing; cluster job scheduling; traffic engineering

Citation: Hariharan P, Chandrasekar N and Padmanabhan P [2022]. Deep Reinforcement Learning for Dynamic Load Balancing in Distributed Computing. **DOI:** <https://doi.org/10.5281/zenodo.22843860>

Copyright: © 2022 by the authors. Open access under [CC BY 4.0 license](https://creativecommons.org/licenses/by/4.0/).

Article Information: Received: 26 February 2022 Accepted: 27 May 2022 Published: 24 August 2022

Article Class: Research Article

1. Introduction

Distributed computing systems, from data-center clusters to wide-area content-delivery networks, all share one recurring operational headache: work arrives unevenly, machines fail or slow down unpredictably, and whatever policy is deciding where the next request or job goes has to make that decision in milliseconds with only a partial, stale view of the system. Load balancing is the name given to that decision problem, and for most of the history of distributed systems it has been solved with policies simple enough to reason about by hand — round robin, least connections, weighted variants of both, or randomized schemes like the power-of-two-choices family. These heuristics are cheap, stateless or nearly so, and well understood, but they share a structural limitation: they react to the system as it looks right now, using a handful of hand-chosen signals, rather than learning from how the system actually behaves over time under the specific, idiosyncratic traffic and failure patterns a given deployment produces.

Reinforcement learning (RL) offers a different starting point. Instead of hand-specifying a rule, RL frames load balancing as a sequential decision problem: an agent observes the state of the system (queue lengths, server utilization, network conditions), takes an action (route this request here, schedule this job there, scale this service up or down), and receives a reward signal (low latency, high throughput, balanced utilization) that it uses to improve its policy over time. Deep reinforcement learning (deep RL) pairs this framework with neural-network function approximation, which matters enormously for load balancing because the state space of a real cluster — dozens to thousands of servers, each with multiple time-varying signals — is far too large for a tabular RL method to handle. The combination, popularized first in game-playing domains through deep Q-networks that reached human-level performance on Atari games from raw pixel input [1], has over the past decade migrated steadily into systems research, where the "game" is now a live cluster and the score is measured in milliseconds of tail latency or watts of energy saved.

This article synthesizes that migration as it applies specifically to dynamic load balancing in distributed computing. We are not reporting a new experiment; we are pulling together what a decade of applied deep RL

research has actually shown about using learned policies to route requests, schedule jobs, and scale services across four layers of the distributed-systems stack: the network layer, where individual packets or connections are load balanced across a fleet of application servers; the cluster-scheduling layer, where jobs compete for shared compute resources; the container-orchestration layer, where microservices are scaled up and down in response to demand; and the wide-area layer, where traffic is routed across an inter-datacenter backbone. Each layer has produced its own applied deep RL literature, and reading them side by side is instructive: the core algorithmic toolkit (value-based methods descending from Q-learning, policy-gradient and actor-critic methods, and increasingly multi-agent formulations) repeats across layers, but the state representations, action spaces, and reward designs that make deep RL actually work at each layer differ substantially, and those differences are where most of the genuine engineering difficulty lives.

We organize the discussion around six themes: the classical load-balancing problem and why static heuristics struggle with dynamic distributed workloads; the reinforcement-learning foundations that make deep RL for load balancing possible at all, from tabular Q-learning through the deep value-based and policy-gradient methods that dominate the applied literature; how the load-balancing problem is formulated as a Markov decision process in practice; deep RL for data-center and network-level load balancing specifically; deep RL for cluster job scheduling, which is a close cousin of load balancing with its own dedicated literature; deep RL for container-orchestration autoscaling; and deep RL for wide-area traffic engineering. We close with a frank discussion of where the literature's evidence is thinner than the enthusiasm around it would suggest, because several of the open problems in this space — evaluation realism, safety during online learning, and multi-agent credit assignment chief among them — are not solved, and treating the field as more mature than it is would do readers a disservice.

2. Classical Load Balancing and the Limits of Static Heuristics

Before deep RL entered the picture, load balancing in distributed systems was, and largely still is in production, handled by a small set of well-understood

algorithms. Round robin cycles through a fixed list of servers regardless of their current load; weighted round robin adjusts the cycle to account for known differences in server capacity; least-connections routes each new request to whichever server currently has the fewest open connections; and randomized load balancing, most famously the "power of two choices" family, samples a small number of servers at random and picks the least-loaded of the sample, which turns out to dramatically outperform naive random assignment at essentially no extra bookkeeping cost. These schemes are attractive for reasons that go beyond simplicity: they require little or no shared state, they degrade gracefully, and their behavior is predictable enough that operators can reason about worst-case outcomes.

Their limitation is that "current load," as any of these heuristics measures it, is a lagging and often noisy indicator of what actually matters, which is how a request placed now will perform over its entire lifetime, possibly seconds or minutes into the future, on a server whose load will itself keep changing. A queue-length snapshot does not distinguish between five short requests and one long-running one; a connection count does not capture whether the server behind that connection is CPU-bound, memory-bound, or waiting on a slow downstream dependency. Heterogeneity compounds the problem: a fleet built up over successive hardware refresh cycles, or a federation spanning edge devices and cloud servers with wildly different capacities, breaks the implicit assumption behind most of these heuristics that one unit of "load" means roughly the same thing on every machine. Google's engineering account of B4, its globally-deployed software-defined wide-area network, is a useful illustration of the stakes involved once you move beyond a single cluster: at WAN scale, static and locally-reactive routing decisions leave substantial bandwidth on the table relative to what a centralized system with a global view of demand and topology can achieve, which is precisely why B4 replaced distributed shortest-path routing with a centralized traffic-engineering controller in the first place [2]. That shift — from local, reactive, hand-specified rules toward centralized systems that optimize against an explicit objective using a fuller view of system state — is also, in a different technical vocabulary, exactly the shift that deep reinforcement learning proposes to make at every layer of the stack, and it is the throughline connecting the rest of this article.

3. Reinforcement Learning Foundations for Sequential Decision-Making

From Tabular Q-Learning to Deep Value-Based Methods

Reinforcement learning's basic object of study is the Markov decision process: at each discrete time step, an agent observes a state, chooses an action, receives a reward, and transitions to a new state, with the goal of learning a policy that maximizes cumulative discounted reward. Watkins and Dayan's Q-learning algorithm, one of the field's foundational results, showed that an agent can learn the optimal action-value function — the expected future reward of taking a given action in a given state and behaving optimally thereafter — purely from sampled transitions, without ever needing a model of the environment's dynamics, and that this estimate converges to the true optimum under mild conditions [3]. For small, discrete state spaces this tabular approach works well, but a real distributed system's state — dozens of servers' worth of queue lengths, CPU utilization, memory pressure, and network latency, all changing continuously — cannot be enumerated in a table.

Deep Q-networks (DQN) solved this by replacing the table with a neural network trained to approximate the action-value function, combined with two stabilization tricks — a replay buffer of past transitions sampled to break temporal correlation, and a separately-updated target network to stabilize the learning target — that together let the approach scale to raw, high-dimensional input and reach human-level performance across a broad suite of Atari games from pixels alone [1]. Two refinements matter for the systems applications discussed later in this article. Van Hasselt, Guez, and Silver showed that standard Q-learning systematically overestimates action values because the same network is used both to select and to evaluate the best action, and proposed Double DQN, which decouples selection from evaluation using the existing online and target networks, yielding more accurate value estimates and more stable policies with essentially no additional computational cost [4]. Wang et al.'s dueling network architecture took a complementary approach, splitting the network into a state-value stream and an action-advantage stream that are recombined into the final action values, which proved particularly helpful in settings where many actions have similar value and the more informative signal is simply how good the current state is,

independent of which action is taken — a pattern that recurs in load-balancing settings where several candidate servers are often nearly interchangeable [5].

Policy-Gradient and Actor-Critic Methods

Value-based methods like DQN learn a value function and derive a policy from it implicitly (act greedily with respect to the estimated values); policy-gradient methods instead parameterize the policy directly and optimize it by gradient ascent on expected reward. This distinction matters for systems applications because many load-balancing and scheduling actions are more naturally continuous (what fraction of traffic to send to this shard) or drawn from very large discrete action spaces (which of a thousand servers should host this job), settings where value-based methods scale poorly. Lillicrap et al.'s deep deterministic policy gradient (DDPG) extended the actor-critic framework to continuous action spaces by learning a deterministic policy network alongside a Q-value critic, using the same replay-buffer and target-network stabilization tricks as DQN, and demonstrated strong performance across a range of continuous-control benchmarks directly from raw state observations [6]. Mnih et al.'s asynchronous advantage actor-critic (A3C) took a different route to stability and sample efficiency, running many actor-learner threads in parallel against independent copies of the environment and asynchronously updating a shared set of parameters, which both stabilizes training (because the parallel experience is naturally decorrelated, removing the need for a replay buffer) and dramatically reduces wall-clock training time on ordinary multi-core hardware [7]. Schulman et al.'s proximal policy optimization (PPO) addressed a persistent practical headache with earlier policy-gradient methods — that a single overly large policy update can catastrophically and irrecoverably degrade performance — by clipping the objective function to discourage the new policy from moving too far from the old one in a single update step, and has since become one of the most widely used RL algorithms in applied work precisely because it combines strong performance with comparatively simple, robust hyperparameter tuning [8]. Nearly every deep RL load-balancing and scheduling system discussed in the remainder of this article builds on some combination of these four algorithmic families: DQN-style value learning, its double and dueling refinements, DDPG-style continuous control, or A3C/PPO-style

policy-gradient learning.

4. Formulating Load Balancing as a Markov Decision Process

Applying any of the algorithms above to load balancing requires committing to a specific state representation, action space, and reward function, and the choices made here vary considerably across the literature in ways that matter more for real performance than the choice of underlying RL algorithm. State representations typically combine some mix of per-server queue length or utilization, recent request or job arrival statistics, network round-trip times, and, in cluster-scheduling settings, the resource profile (CPU, memory, and sometimes GPU demand) of the job waiting to be placed; Mao et al.'s DeepRM system for cluster resource management, one of the earliest and most influential applied deep RL papers in this space, represents the scheduler's state as an image-like grid encoding both current resource allocation and the resource profiles of jobs waiting in the queue, which lets a convolutional policy network reason jointly over allocation and backlog in a single forward pass [9]. Action spaces range from a small discrete choice among candidate servers (well suited to DQN-style value learning) to continuous traffic-splitting ratios across many destinations (better suited to DDPG-style continuous control) to, in the traffic-engineering setting discussed later, a full flow-allocation vector over an entire network topology.

Reward design is where the systems literature departs most visibly from the game-playing origins of deep RL. A game has a single, unambiguous score; a distributed system has multiple, often competing objectives — low latency, high throughput, balanced utilization, minimal job slowdown, low energy consumption — and the reward function has to encode a chosen trade-off among them, usually as a weighted combination or, in more recent multi-objective formulations, as an explicit Pareto-aware objective. Tao and Liu's multi-objective job-scheduling scheme frames this tension directly, treating load balancing and other scheduling objectives (such as makespan or resource utilization) as simultaneous goals a single deep RL policy has to trade off rather than collapsing everything into one scalar reward from the outset [10]. A second recurring design choice is the training regime itself: because training an RL agent directly against a live production cluster risks genuinely bad decisions during the exploration phase,

most of the systems discussed in this article are trained in simulation, against a workload trace or a synthetic generator meant to approximate real conditions, and then either deployed directly or fine-tuned online — a gap between simulated training and production deployment that we return to in the discussion of open challenges below.

5. Deep Reinforcement Learning for Data-Center and Network-Level Load Balancing

The most direct application of deep RL to load balancing operates at the network layer, where a load balancer sits in front of a fleet of application servers and must decide, for every incoming connection or packet flow, which backend to route it to. Yao, Ding, and Clausen's RLB-QMIX system is a particularly instructive case study because it takes multi-agent RL, rather than a single centralized agent, as its starting point: multiple load-balancing agents, each with a partial, local view of traffic and server state, learn cooperative policies using the QMIX algorithm, which factors a joint action-value function into per-agent components under a monotonicity constraint that keeps the whole system trainable via standard value-based methods even though no single agent sees the full state [11]. Evaluated against Shortest Expected Delay, a strong classical heuristic baseline, RLB-QMIX achieved 90th-percentile response times reported as 1.44 times faster than the SED baseline in moderate-scale testing, while also producing a substantially more equitable distribution of load across heterogeneous servers than SED, which the authors report skewed roughly 2.3 times more traffic toward the more powerful servers in the fleet than RLB-QMIX did — a concrete illustration of deep RL's ability to learn fleet heterogeneity implicitly from experience rather than requiring it to be hand-encoded into the routing rule. The same paper is candid about the cost of this: per-flow decisions computed by the learned policy carry measurable additional processing overhead, on the order of a few percent, relative to the classical baselines, a trade-off between decision quality and decision latency that recurs throughout the applied deep RL load-balancing literature.

Closer to the network edge, Esmaeili et al. study dynamic load balancing specifically for edge-computing networks, where the resource-constrained and geographically distributed nature of edge servers makes the classical heuristics' assumption of roughly

homogeneous, centrally visible servers even less appropriate than it is in a conventional data center [12]. Their comparison of three reinforcement-learning formulations — tabular Q-learning, a multi-armed bandit formulation that treats server selection as a stateless repeated-choice problem, and a gradient-bandit variant that learns a preference over servers via stochastic gradient ascent rather than value estimation — is useful precisely because it isolates how much of deep RL's advantage in this setting comes from full sequential decision-making versus a much cheaper stateless learning signal; bandit-style formulations, by discarding the temporal state entirely, are dramatically cheaper to train and deploy, which matters at the edge where the load-balancing decision itself has to run on constrained hardware. Luong et al.'s broad survey of deep RL applications across communications and networking situates both of these lines of work within a much larger literature spanning dynamic spectrum access, network caching, and data offloading, and makes the point, echoed throughout the load-balancing-specific papers surveyed here, that the appeal of deep RL in networking contexts is less about beating hand-tuned heuristics on average performance — well-tuned heuristics are often surprisingly competitive — and more about automatically adapting to workload and topology conditions that were never anticipated at heuristic-design time [13].

Table 1: Representative deep-RL systems for load-related decision-making across the distributed-computing stack, with the effect each cited source reports against its classical baseline. [9,11,12,19,18,17].

Technique	Source	Target Layer	Reported Effect
Policy-gradient job packing (DeepRM)	Mao et al. [9]	Cluster job scheduling	Avg. job slowdown ~3.5 vs ~4.2 for Tetris heuristic at 70% cluster load
Multi-agent QMIX load balancing (RLB-QMIX)	Yao et al. [11]	Data-center network load balancing	1.44x faster 90th-percentile response time than Shortest Expected Delay baseline
Q-learning / bandit load balancing	Esmaeili et al. [12]	Edge-computing networks	Q-learning, multi-armed and gradient bandit policies outperform static load-balancing baselines under dynamic edge conditions
GNN-based deep-RL flow allocation (Teal)	Xu et al. [19]	Wide-area traffic engineering	197-625x faster than acceleration baselines while satisfying 6-32% more traffic demand

Technique	Source	Target Layer	Reported Effect
RL-based container autoscaling (Gwydion)	Santos et al. [18]	Kubernetes multi-service autoscaling	Efficient auto-scaling of complex, interdependent containerized applications validated on a Kubernetes testbed
RL-based microservice autoscaling module	Abdel Khaleq & Ra [17]	Microservice resource scaling	Improved resource utilization over conventional threshold-based autoscaling rules

6. Deep Reinforcement Learning for Job and Task Scheduling Across Clusters

Cluster job scheduling is a close relative of network-level load balancing — both are, at bottom, problems of assigning incoming work to a set of resources under uncertainty about future arrivals — but it operates at a coarser grain (jobs rather than individual packets or connections) and typically over a longer time horizon, which changes both the natural state representation and the reward signal. Mao et al.'s DeepRM system, mentioned above for its state representation, trained a policy-gradient agent to pack variable-sized jobs onto a fixed pool of cluster resources, optimizing directly for average job slowdown (a job's actual completion time relative to what it would have taken running alone with no contention) rather than a proxy like utilization [9]. Against Tetris, a strong multi-resource packing heuristic, and other established scheduling baselines, DeepRM achieved lower average job slowdown across a range of cluster load levels — roughly 3.5 versus approximately 4.2 for Tetris at 70 percent cluster load in the authors' reported experiments — and did so by learning emergent behaviors that were not explicitly programmed, notably a tendency to strategically hold back large jobs to make room for smaller arrivals rather than scheduling strictly in arrival order, a policy the authors observed the learned agent adopting on its own roughly a quarter of the time under high-load conditions.

More recent work has pushed this line of research toward heterogeneous and multi-objective settings that better match real production clusters. Yao et al.'s 2025 study of job scheduling on load-aware heterogeneous clusters extends the DeepRM-style formulation to explicitly account for machines with different capacities and different current loads within the same scheduling

decision, arguing that treating the cluster as an undifferentiated resource pool, as much of the earlier literature implicitly does, systematically under-serves clusters that have grown organically through incremental hardware additions rather than being provisioned uniformly from the start [14]. Tao and Liu's multi-objective scheme, discussed above for its reward formulation, reports that jointly optimizing for load balance alongside conventional scheduling objectives, rather than optimizing scheduling efficiency and then load-balancing as an afterthought, produces schedules that neither single-objective baseline reaches on its own [10]. Zhou et al.'s survey of deep RL-based methods for cloud resource scheduling offers a useful bird's-eye view across this literature, cataloguing the recurring design choices (state representation granularity, single- versus multi-agent formulations, discrete versus continuous action spaces) that distinguish one scheduling paper from the next, and noting that despite substantial diversity in specific technique, the reported gains over well-tuned classical schedulers cluster in a broadly consistent range across studies, which the authors read as evidence that the underlying approach, rather than any single paper's specific architecture, is what is doing the work [15]. Gu et al.'s complementary algorithm-level review reaches a similar conclusion from a different angle, organizing the literature by which specific RL algorithm (value-based, policy-gradient, actor-critic, or multi-agent) each scheduling system builds on, and observing that the choice of underlying algorithm matters less for reported performance than the quality of the state and reward design layered on top of it [16].

7. Deep Reinforcement Learning for Container Orchestration and Microservice Autoscaling

A third, closely related application of deep RL to distributed-systems resource management sits one layer up the stack: rather than deciding which server handles a given request or job, container-orchestration autoscaling decides how many replicas of a service should be running at all, a decision that shapes the load-balancing problem at the layers below it by determining how much capacity is available to balance load across in the first place. Kubernetes' built-in Horizontal Pod Autoscaler makes this decision using simple threshold rules on a small number of metrics, typically CPU or memory utilization, which shares the same fundamental limitation as the classical load-balancing heuristics discussed earlier: a threshold

rule reacts to where utilization is right now, with no notion of where it is heading or what it costs, in scaling delay or resource waste, to react too late or too early. Abdel Khaleq and Ra's reinforcement-learning-based microservices autoscaling module directly targets this gap, learning a scaling policy from observed workload patterns rather than a fixed threshold, and reporting improved resource utilization relative to conventional threshold-based autoscaling rules across the microservice workloads they evaluated [17]. Santos et al.'s Gwydion system pushes this further toward the complexity of real production topologies, targeting autoscaling for complex containerized applications composed of multiple interdependent microservices, where a scaling decision for one service has downstream effects on the load its dependent services need to absorb, and validating a reinforcement-learning autoscaler against this multi-service interdependency directly on a Kubernetes testbed rather than a single-service simplification of the problem [18].

The relevance of this layer to load balancing proper is sometimes underappreciated in discussions that treat load balancing and autoscaling as separate concerns. A load balancer, however cleverly it distributes traffic, cannot compensate for a fleet that is simply undersized for current demand, and an autoscaler that reacts too slowly leaves the load balancer distributing traffic evenly across a fleet that is, as a whole, overloaded. The two problems are best understood as coupled layers of the same underlying resource-management question, and the fact that deep RL has been applied productively to both, using substantially the same algorithmic toolkit, is one of the more coherent threads running through this literature.

8. Wide-Area Traffic Engineering with Deep Reinforcement Learning

The final layer we consider moves outside a single data center entirely, to the problem of routing traffic across the wide-area network links connecting geographically distributed data centers — the same problem Google's B4 system was built to address with centralized, non-learned optimization [2]. Wide-area traffic engineering (TE) is computationally demanding in a way the other layers discussed so far are not: a TE controller has to solve, in near real time, an optimization problem over the entire network's flow allocation, subject to link-capacity constraints and a global objective like

maximizing total satisfied demand, and the classical approach — formulating this as a linear or convex program and solving it exactly with an off-the-shelf solver — scales poorly as networks and traffic matrices grow, because the underlying optimization problem's size grows with the number of flows and links, not just the number of decisions being made per flow.

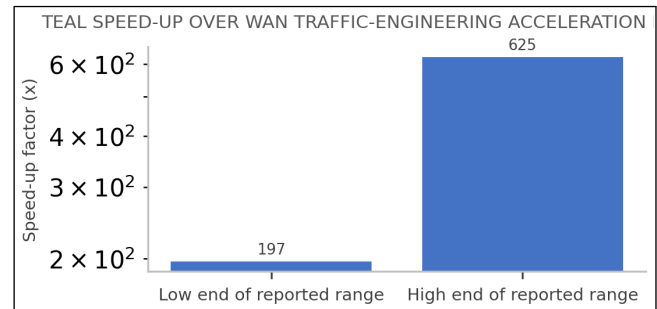


Figure 1: Range of reported speed-ups of Xu et al.'s Teal deep-RL traffic-engineering system over competing WAN acceleration baselines on the same flow-allocation problem, while simultaneously satisfying 6-32% more traffic demand -- an unusually rare case in this literature of a learned policy improving both solution quality and decision latency at once [19].

Xu et al.'s Teal system reframes this optimization problem as a learning problem: rather than solving the flow-allocation optimization from scratch on every TE update, a graph-neural-network-based deep RL policy learns to produce near-optimal flow allocations directly, using the network topology and current traffic demand as input, and is subsequently fine-tuned with a small number of gradient steps toward exact feasibility rather than solved from a blank slate [19]. Evaluated against a large wide-area network topology using real Microsoft traffic matrices, Teal is reported to run several orders of magnitude faster than the production optimization engine it was compared against, and, relative to other acceleration schemes for the same underlying TE problem, to both satisfy 6 to 32 percent more traffic demand and deliver speedups ranging from roughly 197 to 625 times, depending on the specific baseline and network scale in the comparison. That combination — better solution quality and dramatically lower latency simultaneously, rather than the more typical trade-off between the two — is a notable result within the broader deep RL for load-balancing and resource-management literature, most of which reports modest quality improvements at some added computational cost rather than improvements on both axes at once, and it is a large part of why Teal has become a frequently cited reference point for what learning-accelerated network optimization can achieve. Cheng et al.'s

reinforcement-learning-based traffic-engineering algorithm for enterprise backbone links tackles a related but more modest-scale version of the same problem, incorporating both the priority weighting and traffic classification of individual flows into the learned routing policy, and is a useful reminder that wide-area traffic engineering is not exclusively a hyperscale-operator concern; the same fundamental problem, at a smaller scale, shows up in any organization operating multiple sites connected by backbone links whose capacity is worth using efficiently [20].

9. Open Challenges

Several limitations recur across the literature synthesized here and deserve to be stated plainly. First, evaluation realism remains the field's most persistent weakness. The large majority of the systems discussed above, including DeepRM, RLB-QMIX, and the microservice-autoscaling systems, are evaluated in simulation or on constrained testbeds against synthetic or trace-replayed workloads rather than genuinely live, adversarial production traffic, and while simulation is a reasonable and often necessary starting point for training an RL agent safely, the gap between a workload trace generated for research purposes and the messiness of real production traffic — correlated bursts, adversarial or simply unusual client behavior, hardware failures that do not appear in a trace at all — is large and, across this literature, inconsistently characterized [9,11,17]. Second, and closely related, is the safety of online learning: a load-balancing or autoscaling policy that continues to adapt after deployment is, by construction, capable of making a genuinely bad decision it has not encountered during training, and relatively few of the systems surveyed here specify a concrete mechanism — a fallback heuristic, a bounded action space, a conservative exploration schedule — for containing the damage a bad online update could cause in production, as opposed to simply reporting strong average-case performance in evaluation.

Third, multi-agent formulations like RLB-QMIX raise credit-assignment and coordination questions that single-agent RL does not have to answer at all: when several load-balancing agents act on local, partial views of a shared system, it is not obvious how to attribute a good or bad outcome to any individual agent's decision, and the monotonicity assumptions that make value-decomposition methods like QMIX tractable are a

real constraint on what kinds of coordinated behavior the resulting policies can learn, not merely an implementation detail [11]. Fourth, reward design remains as much art as science across this literature: the shift from single-objective formulations toward explicit multi-objective rewards, as in Tao and Liu's work, is a genuine improvement in honesty about what these systems are actually being asked to optimize, but the field has not converged on standard practice for how competing objectives like latency, fairness, and energy consumption should be weighted against each other, and different papers make this choice differently with limited justification beyond "it worked well in our experiments" [10]. Fifth, benchmarking and reproducibility remain inconsistent: reported performance gains are expressed in whatever units and against whatever baselines each individual paper chose, from percentage reductions in 90th-percentile latency to multiplicative speedups against a specific production system, which makes it genuinely difficult for a practitioner to compare across papers or predict how a given technique would perform on their own infrastructure without reimplementing and re-testing it directly. None of these gaps make the case against deep RL for load balancing as an architectural pattern — the diversity of layers at which independent research groups have successfully applied broadly similar techniques is itself meaningful evidence that something real is being captured — but they are the honest state of the field, and claims of production-readiness for any specific technique surveyed here should be read with these caveats attached.

10. Conclusion

The literature surveyed in this article supports a fairly specific claim: deep reinforcement learning is a genuinely productive framework for dynamic load balancing across essentially every layer of the distributed-computing stack examined here — network-level connection routing [11,12], cluster job scheduling [9,14,10], container-orchestration autoscaling [17,18], and wide-area traffic engineering [19,20] — precisely because the common structure underlying all four problems, a sequential decision made under uncertainty about future arrivals and system state, is exactly what reinforcement learning was designed to handle. The algorithmic toolkit doing the work is not exotic: value-based methods descending from Q-learning and DQN [3,1,4,5], policy-gradient and

actor-critic methods including DDPG, A3C, and PPO [6,7,8], and increasingly multi-agent extensions of both, applied with problem-specific state representations, action spaces, and reward functions that reflect real understanding of the systems problem being solved rather than a naive transplant from game-playing benchmarks.

What the literature does not yet support is a claim of maturity. The evaluation gap between simulated or testbed studies and adversarial production conditions, the underspecified safety story for policies that keep learning after deployment, the coordination difficulties inherent to multi-agent formulations, and the lack of standardized benchmarking practice are all real, and a fair reading of this body of work treats deep RL for load balancing as a demonstrably promising architectural pattern with a genuine, if uneven, track record — not as a solved problem ready for unreflective production adoption. The consistency with which independent groups working on network load balancing, cluster scheduling, container autoscaling, and wide-area traffic engineering have converged on structurally similar approaches, and reported meaningful, verifiable improvements over well-established classical baselines, is nonetheless the strongest available signal that this convergence reflects something durable about the underlying problem rather than a passing research fashion, and it is reason enough to expect deep reinforcement learning to remain a central tool for dynamic load balancing in distributed computing for the foreseeable future.

11. References

- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., et al. (2015) Human-Level Control through Deep Reinforcement Learning. *Nature*, 518(7540): 529-533. DOI: [10.1038/nature14236](https://doi.org/10.1038/nature14236)
- Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., Vahdat, A. (2013) B4: Experience with a Globally-Deployed Software Defined WAN. *Proceedings of the ACM SIGCOMM 2013 Conference*, 3-14. DOI: [10.1145/2486001.2486019](https://doi.org/10.1145/2486001.2486019)
- Watkins, C.J.C.H., Dayan, P. (1992) Q-Learning. *Machine Learning*, 8(3-4): 279-292. DOI: [10.1007/BF00992698](https://doi.org/10.1007/BF00992698)
- van Hasselt, H., Guez, A., Silver, D. (2016) Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1). DOI: [10.48550/arXiv.1509.06461](https://doi.org/10.48550/arXiv.1509.06461)
- Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., de Freitas, N. (2016) Dueling Network Architectures for Deep Reinforcement Learning. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, PMLR 48. DOI: [10.48550/arXiv.1511.06581](https://doi.org/10.48550/arXiv.1511.06581)
- Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D. (2016) Continuous Control with Deep Reinforcement Learning. *International Conference on Learning Representations (ICLR)*. DOI: [10.48550/arXiv.1509.02971](https://doi.org/10.48550/arXiv.1509.02971)
- Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K. (2016) Asynchronous Methods for Deep Reinforcement Learning. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, PMLR 48. DOI: [10.48550/arXiv.1602.01783](https://doi.org/10.48550/arXiv.1602.01783)
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017) Proximal Policy Optimization Algorithms. *arXiv preprint*. DOI: [10.48550/arXiv.1707.06347](https://doi.org/10.48550/arXiv.1707.06347)
- Mao, H., Alizadeh, M., Menache, I., Kandula, S. (2016) Resource Management with Deep Reinforcement Learning. *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets-XV)*, 50-56. DOI: [10.1145/3005745.3005750](https://doi.org/10.1145/3005745.3005750)
- Tao, Y., Liu, J. (2026) Deep Reinforcement Learning-Based Multi-Objective Job Scheduling Scheme for Load Balancing. *Cluster Computing*, 29. DOI: [10.1007/s10586-026-05979-0](https://doi.org/10.1007/s10586-026-05979-0)
- Yao, Z., Ding, Z., Clausen, T. (2022) Multi-Agent Reinforcement Learning for Network Load Balancing in Data Center. *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM)*. DOI: [10.1145/3511808.3557133](https://doi.org/10.1145/3511808.3557133)
- Esmaili, M.E., Khonsari, A., Sohrabi, V., Dadlani, A. (2024) Reinforcement Learning-Based Dynamic Load Balancing in Edge Computing Networks. *Computer Communications*, 222: 188-197. DOI: [10.1016/j.comcom.2024.04.009](https://doi.org/10.1016/j.comcom.2024.04.009)
- Luong, N.C., Hoang, D.T., Gong, S., Niyato, D., Wang, P., Liang, Y.C., Kim, D.I. (2019) Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Communications Surveys & Tutorials*, 21(4): 3133-3174. DOI: [10.1109/COMST.2019.2916583](https://doi.org/10.1109/COMST.2019.2916583)
- Yao, Z., Ding, L., Zhang, H., Li, H., Chen, L., Li, Z. (2025) Deep Reinforcement Learning for Job Scheduling on Load-Aware Heterogeneous Cluster. *Journal of Big Data*, 12: 276. DOI: [10.1186/s40537-025-01325-0](https://doi.org/10.1186/s40537-025-01325-0)
- Zhou, G., Tian, W., Buyya, R., Xue, R., Song, L. (2024) Deep Reinforcement Learning-Based Methods for Resource Scheduling in Cloud Computing: A Review and Future

Directions. Artificial Intelligence Review, 57(5): 124. DOI: [10.1007/s10462-024-10756-9](https://doi.org/10.1007/s10462-024-10756-9)

16. Gu, Y., Liu, Z., Dai, S., Liu, C., Wang, Y., Wang, S., Theodoropoulos, G., Cheng, L. (2025) Deep Reinforcement Learning for Job Scheduling and Resource Management in Cloud Computing: An Algorithm-Level Review. arXiv preprint. DOI: [10.48550/arXiv.2501.01007](https://doi.org/10.48550/arXiv.2501.01007)
17. Abdel Khaleq, A., Ra, I. (2023) Intelligent Microservices Autoscaling Module Using Reinforcement Learning. Cluster Computing, 26(5): 2789-2800. DOI: [10.1007/s10586-023-03999-8](https://doi.org/10.1007/s10586-023-03999-8)
18. Santos, J., Reppas, E., Wauters, T., Volckaert, B., De Turck, F. (2025) Gwydion: Efficient Auto-Scaling for Complex Containerized Applications in Kubernetes through Reinforcement Learning. Journal of Network and Computer Applications, 234: 104067. DOI: [10.1016/j.jnca.2024.104067](https://doi.org/10.1016/j.jnca.2024.104067)
19. Xu, Z., Yan, F.Y., Singh, R., Chiu, J.T., Rush, A.M., Yu, M. (2023) Teal: Learning-Accelerated Optimization of WAN Traffic Engineering. Proceedings of the ACM SIGCOMM 2023 Conference. DOI: [10.48550/arXiv.2210.13763](https://doi.org/10.48550/arXiv.2210.13763)
20. Cheng, H., Luo, Y., Zhang, L., Liao, Z. (2024) A Reinforcement Learning-Based Traffic Engineering Algorithm for Enterprise Network Backbone Links. Electronics, 13(8): 1441. DOI: [10.3390/electronics13081441](https://doi.org/10.3390/electronics13081441)

12. Declarations

Funding

This literature synthesis was conducted without a specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Conflict of Interest

The authors declare that they have no competing interests relevant to the content of this article.

Data Availability Statement

This is a literature synthesis and no new data were generated or analysed. All findings discussed are drawn from the cited sources, which readers should consult for their own data and code availability statements.

Ethical Approval

Not applicable. This work is a synthesis of published literature and did not involve human participants, animal subjects, or the collection of primary data.