EASQ: Energy-Aware Container Scheduling with Tabular Q-Learning for Orchestrated Cloud Clusters

Vivek Saxena* and Anoop Kumar

Department of Computer Science and Engineering, Banasthali Vidyapith, Rajasthan, India
{viveks.mec@gmail.com*, anupbhola@banasthali.in}

Abstract. Default container schedulers place workloads using rules fixed at deployment time: Docker Swarm's spread strategy equalises container counts across nodes, and Kubernetes's bin-packing (MostAllocated) scoring profile packs them by requested—not consumed—resources. Both ignore what a placement costs in watts. We formulate container placement as a Markov decision process and train a tabular Q-learning agent, EASQ, whose shaped reward charges each placement its *marginal power increase* while penalising utilisation beyond a safety threshold. A rank-based action space and a coarse 80-state encoding keep the learned policy small enough (640 Q-values) to inspect by hand. On a simulated eight-node heterogeneous cluster driven by diurnal workloads of roughly 2,000 containers per day, EASQ consumed 23.43 ± 0.77 kWh/day across five unseen traces—27.7% below the Swarm-style spread policy and 5.7% below best-fit bin packing—while recording zero CPU overload samples, against 41 per day for best-fit. The advantage is not uniform: best-fit strands only 21.4% of powered-on capacity versus 40.3% for EASQ, and with only five evaluation seeds the Wilcoxon signed-rank test cannot fall below $p = 0.0625$. The previous design that used a delayed cluster-level reward completely failed, reaching only 30.3 kWh/day. The switch to per-action marginal-cost shaping is what made this method work. Ablations 3.95 kWh/day to the packing term and all overload protection to the threshold penalty. Whether these gains hold in a real-world Kubernetes cluster remains to be demonstrated and is the immediate next step of this doctoral work.

Keywords: Container orchestration · Reinforcement learning · Energy-efficient scheduling · Kubernetes · Docker Swarm · Green computing · Q-learning.

1 Introduction

A Kubernetes pod requesting 1,000 millicores and using 350 still blocks the entire 1,000 from all other tenants. This disconnect between requested and consumed capacity, when extrapolated across a cluster, is the principal driver of stranded resources in production container platforms: measurement studies of large fleets report mean CPU utilization between 20% and 40% even with mature autoscaling [1, 2]. Idle capacity is not free. For example, a server that consumes 70W at idle and 220W at full load (30% utilization) spends most of its power budget doing nothing useful [3, 4]. Hence, every node powered unnecessarily turns allocation slack into wasted energy.

© The Author(s) 2026
A. Shastri et al. (eds.), *Proceedings of the Fourth Workshop on Applications of Computational Intelligence and Interactive Systems (WACIIS 2026)*, Advances in Intelligent Systems Research 217,
https://doi.org/10.2991/978-94-6239-832-0_11

Default schedulers make this worse in different ways. Docker Swarm's spread strategy equalises *container counts* across nodes, which keeps every machine powered on regardless of demand [5, 6]. The Kubernetes default scheduler filters and scores nodes using *requested* resources and static weights [7, 8]; its bin-packing variant (the MostAllocated scoring profile) consolidates aggressively but, because it never observes actual consumption, it happily stacks containers whose true usage exceeds node capacity. Neither policy adapts when the workload mix changes at runtime.

Reinforcement learning offers a mechanism for such adaptation: treat each placement as an action, observe the resulting utilisation and power, and improve the policy from that feedback [9–11]. Prior RL schedulers, however, mostly target job latency or cost [10, 12, 13], use deep networks whose decisions are hard to audit in an operations context [14], or validate on homogeneous clusters where the energy question is less interesting. Our earlier workshop paper [15] sketched the architectural concept behind this work but contained no formalisation and no experiments; the present paper supplies both.

We make four contributions, one of which is negative. We formalise energy-aware container placement as an MDP whose action space is a node's *rank* in the utilisation ordering rather than its identity, cutting the state–action table to 640 entries on an eight-node cluster (Sect. 4). We introduce a marginal-cost shaped reward that charges each placement the instantaneous power increase it causes—including the idle-power step of waking a sleeping node—and show in an ablation that removing any of its three terms costs either energy (up to $+3.95$ kWh/day) or safety (28 overload samples/day) (Sect. 6). We report a failed first design: a delayed, cluster-averaged reward that reviewers might plausibly have expected to work trained to only 30.3 kWh/day, barely ahead of least-loaded placement, and we explain mechanistically why its credit assignment breaks (Sect. 4.3). All simulator code, traces, and evaluation seeds are released so the numbers in every table can be regenerated from a single script.

The evaluation is simulation-based. We state this plainly because it bounds the claims: the cluster, workload, and power models (Sect. 5) are parameterised from published measurements [4, 18, 1], but no simulation substitutes for a physical testbed, and Sect. 7 quantifies what could change on real hardware.

2 Related Work

2.1 Orchestrator Placement Policies and Their Blind Spots

Comparative studies of Swarm and Kubernetes agree on the mechanics: Swarm propagates via number of containers, Kubernetes filters by requests and limits [5, 16]. Benchmarking work showed that containers are more efficient than virtual machines in CPU, memory and I/O [17–19]. This is why the energy result of a cluster is now mainly driven by container-level (not VM-level) scheduling decisions. What this literature largely assumes is the common assumption that requested resources are a usable proxy for

demand. Google's Borg data indicated that requests overestimated usage by 40-60 % at fleet scale [2, 1]; any scorer built on requests inherits that error at every decision.

2.2 Optimisation- and Learning-Based Scheduling

Metaheuristics—tabu search with Bayesian classification, whale optimisation [13], K-means-driven allocation [20]—improve cost or makespan but solve a static snapshot; the schedule is recomputed wholesale when conditions drift. Learning-based work moved the adaptation online. DeepRM framed job scheduling as deep RL over resource images [10]; later systems applied prediction [12] and RL [11] to cloud allocation, and surveys of resource-consumption optimisation catalogue dozens of variants [21, 22]. Two gaps persist across this family. Energy rarely enters the reward directly—latency and cost dominate—and when deep networks are used, the resulting policy resists the kind of line-by-line audit that cluster operators demand before ceding placement control. Jang et al.'s virtual-CPU scheduler [23] is closer in spirit, optimising CPU efficiency inside the hypervisor, but it operates below the orchestration layer and cannot exploit cluster-level signals such as which nodes could sleep.

2.3 Energy-Aware Resource Management

Energy-proportional computing [3] and the linear power–utilisation model validated on warehouse fleets [4] underpin most energy-aware allocators, including the VM-consolidation heuristics of Beloglazov and Buyya [24]. Consolidation-versus-reliability tension is well documented there: packing tighter saves idle power until a demand spike overruns a node. The Green AI agenda [25, 26] adds a second motivation—making energy a first-class objective of learned systems, not an afterthought. Our reading of these three threads is that the container-orchestration layer is where they have failed to meet: consolidation logic exists for VMs, learned policies exist for latency, but a small, auditable, energy-rewarded learner for container placement had, to the best of our knowledge, not been evaluated with its failure modes reported. Filling exactly that slot—with the caveats such a claim deserves—is the aim here.

3 System Model and Problem Formulation

3.1 Cluster and Workload Model

A cluster consists of M heterogeneous nodes. Node j has CPU capacity C_j (millicores), memory capacity R_j (MiB), idle power P_j^{idle}, and peak power $P_j^{\max}$. Containers arrive as a marked point process: container i arrives at time t_i with CPU request c_i, memory request m_i, service duration d_i, and a latent usage ratio $\rho_i \in (0,1]$ unknown to the scheduler. Its instantaneous consumption on its host is

$$u_i(t) = c_i\,\rho_i(1 + 0.15\sin(0.13\,t + \phi_i)), \tag{1}$$

where the sinusoidal term with random phase ϕ_i injects the short-period fluctuation observed in per-container CPU traces [1]. The multiplier 0.15 was chosen to match the coefficient of variation reported for Google trace tasks; larger values mainly widen the overload counts of aggressive packers.

Node j's utilisation and the admission constraints are

$$u_j(t) = \frac{1}{C_j}\sum_{i\in\mathcal{A}_j(t)} u_i\,(t), \qquad \sum_{i\in\mathcal{A}_j(t)} m_i \le R_j, \qquad \sum_{i\in\mathcal{A}_j(t)} c_i \le 2\,C_j, \tag{2}$$

with $\mathcal{A}_j(t)$ the set of containers on j. Memory is a hard constraint; CPU admits 2× overcommit, mirroring common Kubernetes practice where limits exceed requests. Utilisation above 1.0 is possible under overcommit and models throttling: we count each 5-minute sample with $u_j > 1$ as an *overload event*, our proxy for SLO risk.

3.2 Energy Model

Before stating the power equation we note why its form matters to the learning problem. Because idle power is a large fraction of peak—roughly 32–38% for our node classes, consistent with published server measurements [4, 3]—the power cost of a placement is dominated by *whether it wakes a node*, not by the marginal utilisation it adds to an already-active one. Any reward that masks this step function from the agent throws away the main signal. Node power is modeled in the standard linear way with a sleep state:

$$P_j(t) = \begin{cases} P^{\text{sleep}}, & \mathcal{A}_j(t) = \emptyset, \\ P_j^{\text{idle}} + (P_j^{\max} - P_j^{\text{idle}})\,\min(u_j(t),1), & \text{otherwise}, \end{cases} \tag{3}$$

and cluster energy over horizon T is $E = \int_0^T \sum_{j=1}^{M} P_j\,(t)\,dt$. In plain terms: an empty node costs $P^{\text{sleep}} = 10$ W, the first container on it costs the full idle step, and each further unit of load costs only the difference between peak and idle, spread over capacity.

3.3 Scheduling as a Markov Decision Process

Each container arrival triggers one decision. The scheduler observes state s_t, chooses a node (action a_t), receives reward r_t, and the system transitions as containers arrive and depart. The objective is a policy π maximising expected discounted return $\mathbb{E}_\pi[\sum_k \gamma^k r_{t+k}]$, which we shape (Sect. 4.2) so that maximising return simultaneously suppresses energy and overload risk. Decisions interact—parking a container on a nearly-full node now may force waking a node later—which is why we retain the sequential formulation rather than treating each placement greedily, although the sensitivity analysis in Sect. 6.4 shows the effective horizon that helps is shorter than we expected.

4 The EASQ Scheduler

4.1 Compact State via Rank-Based Actions

A naive state (per-node utilisation levels) explodes combinatorially: eight nodes at five levels each is $5^8 \approx 390$k states before adding container features. EASQ sidesteps this with two moves. Node identity is replaced by *rank*: at decision time the nodes are sorted by current utilisation, and the action $a \in \{0, \dots, M-1\}$ selects a position in that ordering (0 = emptiest node). Rank captures what matters for consolidation, i.e. "pack the fullest feasible node" is rank M-1, but is invariant to node relabeling. Then only cluster level statistics needs:

$$s_t = (g_i,\ \mathrm{bin}_5(\bar{u}_t),\ \mathrm{bin}_4(\sigma_{u,t})), \tag{4}$$

where $g_i \in \{1, \dots, 4\}$ is the arriving container's size class, $\bar{u}_t$ and $\sigma_{u,t}$ are the mean and standard deviation of node utilisations (bin edges $\{0.15, 0.30, 0.45, 0.60\}$ and $\{0.08, 0.16, 0.28\}$), giving $4 \times 5 \times 4 = 80$ states and a 80×8 Q-table. Every entry can be printed on one page—an auditability property no deep policy offers. The cost of this compression is real: rank-based actions cannot express preferences between two nodes of equal utilisation but different power efficiency, a limitation we return to in Sect. 7.

The rank-based encoding keeps the state space fixed at 80 cells regardless of cluster size, but the action space assigns one action per rank, so the Q-table has shape 80×M and its column dimension grows with the number of nodes M; a table trained for one cluster size cannot be reused as-is for another and must be retrained from scratch whenever M changes. Because the state features already abstract away node identity, we would not expect retraining time to grow faster than the larger table needing more visits per cell to reach comparable estimates, but we have not tested clusters larger than the eight nodes used here, and adding node classes beyond the three evaluated would likely require finer utilisation bins to keep rank a meaningful signal. We treat this scaling behaviour as an open empirical question rather than a demonstrated property of the design (Sect. 7).

Infeasible ranks (nodes failing Eq. (2)) are masked at both training and inference time; if no node is feasible the container is rejected (this never occurred in evaluation; see Table 2).

4.2 Marginal-Cost Shaped Reward

The reward for placing container i on the node at rank a (call it j, with post-placement utilisation u_j') is computed immediately from measurable quantities:

$$r_t = w_u \min(u_j', \theta) \ - \ w_e \frac{\Delta P_t}{P_{\mathrm{ref}}} \ - \ 5\, w_o \max(0,\ u_j' - \theta), \tag{5}$$

with ΔP_t the cluster power increase caused by this placement (Eq. 3 evaluated before and after), $P_{\mathrm{ref}} = 100\,\mathrm{W}$ a normaliser, $\theta = 0.85$ a safety threshold, and weights $(w_u, w_e, w_o) = (1,1,2)$. Reading the three terms left to right: the first pays for

packing—utilisation on the chosen node is rewarded, but only up to θ, so there is no incentive to overload; the second charges the placement its marginal wattage, which is small on an active node and large (the idle step) when a sleeping node is woken; the third turns the region above θ from "no further reward" into an explicit penalty whose slope (factor 5) makes threshold violations costlier than any packing gain. Q-values are updated by the standard temporal-difference rule [27, 9]

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)], \tag{6}$$

where s_{t+1} is the state at the *next arrival*, not the next simulator tick—decision epochs are the natural clock here.

Several operational constants were fixed by domain convention rather than tuned, and we clarify their basis here since the sensitivity analysis in Sect. 6.4 sweeps only the learning hyperparameters α and γ. The 2× CPU overcommit ratio mirrors the limits-over-requests headroom documented for Kubernetes resource management [7] rather than a value we searched over. The P^{sleep} = 10 W sleep-state power reflects typical ACPI standby draw for the commodity server classes in Table 1 rather than the active idle power. The safety threshold θ = 0.85 was set below saturation by the same margin common in autoscaling target-utilisation policies so that the packing term saturates before overcommit risk becomes material; we did not sweep θ. The reward weights (w_u, w_e, w_o) = (1, 1, 2) were fixed before any evaluation-seed results were inspected, giving the overload penalty double weight on the reasoning that a throttling event is costlier to an operator than an equivalent amount of stranded capacity. The ablation in Sect. 6.3 shows the qualitative role of each reward term but not the effect of alternative weight ratios; extending the sensitivity study to these four constants, not only α and γ, is a natural next step.

4.3 The Design That Did Not Work

Our first reward was the one that seemed principled: a delayed, cluster-level signal $r = w_u \bar{u}^{\text{act}} - w_e P/P^{\max} - w_o f^{\text{over}}$ evaluated at the next decision epoch, where $\bar{u}^{\text{act}}$ averages utilisation over active nodes and f^{over} is the overloaded fraction. Trained identically (120 episodes, same $\alpha, \gamma, \varepsilon$ schedule), it evaluated at 30.33 ± 0.42 kWh/day—worse than best-fit by 5.5 kWh and only 0.6 kWh ahead of least-loaded. The failure is a credit-assignment problem, not a tuning problem: between two consecutive arrivals the cluster-level quantities move mostly because of workload dynamics (arrivals elsewhere, departures, usage jitter), so the correlation between an individual action and its reward is weak, and with ∼2,000 noisy updates per episode spread over 640 table cells, the packing signal drowns. Switching to Eq. 5, which isolates exactly the consequence the action controls, moved evaluation energy from 30.33 to 23.43 kWh/day with no other change. We report this because the delayed variant is the formulation a reader would most likely reproduce first.

4.4 Algorithms and Complexity

Algorithms 1–2 specify placement and training; inference reuses Algorithm 1 with $\varepsilon = 0$. Per decision, sorting dominates: $O(M \log M)$ time and $O(|\mathcal{S}| \cdot M)$ memory for the table (640 floats here). The placement decision on the evaluation hardware (2-core x86 VM) took well under a millisecond and full training (120 day-length episodes, $\sim$240k decisions) was completed in about 2.2 minutes, a budget compatible with periodic retraining on a live cluster.

Require: container (c_i, m_i, g_i); node states $\{(u_j, C_j, R_j)\}_{j=1}^{M}$; table Q; exploration rate ε
Ensure: selected node $j^\star$ or REJECT

$\pi \leftarrow \text{argsort}(u_1, \ldots, u_M)$ ▷ ascending utilisation; $\pi(a)$ = node at rank a
$\mathcal{F} \leftarrow$ {a:node $\pi(a)$ satisfies memory and overcommit bounds of Eq. (2)}
if $\mathcal{F} = \emptyset$ **then return** REJECT
$s \leftarrow (g_i, \text{bin}_5(\bar{u}), \text{bin}_4(\sigma_u))$ ▷ Eq. (4)
if rand() $< \varepsilon$ **then** ▷ explore
  $a \leftarrow$ uniform sample from $\mathcal{F}$
else ▷ exploit, masked
  $a \leftarrow \text{argmax}_{a' \in \mathcal{F}} Q(s, a')$
end if
return $j^\star = \pi(a)$, remembering (s, a) for the pending update

Algorithm 1. EASQ rank-based placement (one decision)

Require: training traces $\{\mathcal{T}_e\}_{e=1}^{E}$; rates α, γ; schedule $\varepsilon_0, \varepsilon_{\min}, \kappa$
Ensure: trained table Q

$Q \leftarrow \mathbf{0}$; $\varepsilon \leftarrow \varepsilon_0$
for episode $e = 1$ **to** E **do**
  reset cluster; PENDING $\leftarrow \varnothing$
  for all container arrivals i in $\mathcal{T}_e$ (time order) **do**
    $s' \leftarrow$ current state via Eq. (4)
    if PENDING $= (s, a, r)$ **then** ▷ delayed one-step update
      $Q(s,a) \leftarrow Q(s,a) + \alpha\,[\, r + \gamma \max_{a'} Q(s', a') - Q(s,a)\,]$
    end if
    run Algorithm 1 → node $j^\star$, pair (s, a)
    if not REJECT **then**
      $P^- \leftarrow \sum_j P_j$; place i on $j^\star$; $P^+ \leftarrow \sum_j P_j$ ▷ Eq. (3)

$r \leftarrow w_u \min(u_{j^\star}', \theta) - w_e(P^+ - P^-)/P_{\text{ref}} - 5w_o \max(0, u_{j^\star}' - \theta)$
$\text{PENDING} \leftarrow (s, a, r)$
end if
end for
$\varepsilon \leftarrow \max(\varepsilon_{\min}, \kappa\varepsilon)$
end for
return Q

Algorithm 2. EASQ training with marginal-power reward shaping

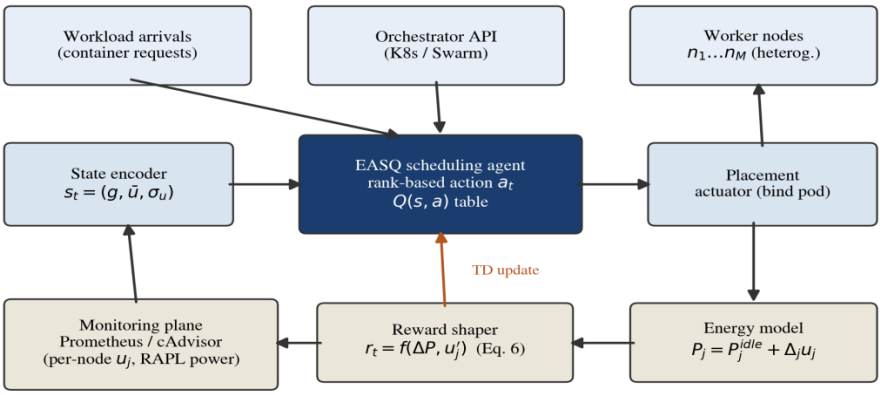

Fig. 1. EASQ closes a loop that is absent in the regular schedulers: the monitoring system provides the recorded utilization and power to the state encoder and reward shaper, while temporal-difference updates change the placement strategy in real-time. The contribution is the dark module and reward pathway. Conventional implementations have shaded elements .

5 Experimental Setup

5.1 Simulator, Cluster, and Workload

We evaluate in a discrete-time simulator (1-minute resolution, 24-hour episodes) built for this study. Simulation is standard for scheduler research where controlled repetition matters [28]; every number below is the direct output of the released code. Table 1 describes the cluster: 8 nodes in 3 classes, where idle/peak powers are based on

published server ranges [4]. The energy question degenerates on identical nodes spread and least-loaded coincide, heterogeneity is by design.

.

Table 1. Simulated cluster (top) and workload (bottom) configuration.

Node class	Count	CPU / memory	Power idle–peak (sleep 10 W)
Small	4	4,000 m / 8 GiB	70–220 W
Medium	2	8,000 m / 16 GiB	110–320 W
Large	2	16,000 m / 32 GiB	170–450 W

Workload parameters (Table 1, bottom) yield ≈2,060 containers per day with a mid-afternoon peak. The Beta(2,3) usage ratio (mean 0.4) encodes the 40–60% over-request behaviour of production fleets [2, 1]. Traces are seeded: training uses seeds 50–59 (fresh trace per episode), evaluation uses five disjoint seeds 0–4 never touched during training or tuning.

5.2 Baselines, Metrics, and Protocol

Four baselines cover the deployed policy space: *Spread* (fewest containers first; Swarm default [6]), *Best-Fit* (highest post-placement requested allocation; Kubernetes MostAllocated [7]), *Round-Robin*, and *Least-Loaded* (lowest measured utilisation; the strongest overload-avoider). These four are representative, deliberately configured policies rather than exhaustive descriptions of what Swarm or Kubernetes can do: Swarm also ships binpack and random strategies, and Kubernetes scoring plugins can be reweighted or replaced away from MostAllocated, so "Swarm-style spread" and "Kubernetes-style bin packing" should be read as the specific configurations named above, not as fixed platform behaviour. We report per day: cluster energy (kWh), mean utilisation of active nodes, stranded capacity (powered-on but unused CPU as % of total), overload samples ($u_j > 1$ at 5-min sampling), utilisation imbalance (std. across nodes), and rejections. EASQ trains for 120 episodes ($\alpha = 0.1, \gamma = 0.95, \varepsilon$: $1.0 \rightarrow 0.05$, decay 0.985/episode) and is evaluated greedily. All results are mean $\pm$ std over the five evaluation seeds; paired Wilcoxon signed-rank tests compare EASQ against each baseline on the same traces.

6 Results and Discussion

6.1 Convergence

Training return climbs throughout the budget (Fig. 2): the 10-episode moving average rises from ≈825 at episode 40 to ≈963 at episode 80 and ≈1,010 by episode 110, with per-episode returns in the final stretch ranging 1,000–1,080. Two effects overlap in that

late improvement and we cannot fully separate them—continued Q-value refinement, and the exploration rate still decaying (it ends at 0.16, above its 0.05 floor, so roughly one placement in six remains random even in the last episode). Greedy evaluation sidesteps the second effect, but a longer schedule might extract a little more; we ran 120 episodes because the wall-clock budget (about two minutes) already suited the intended periodic-retraining deployment.

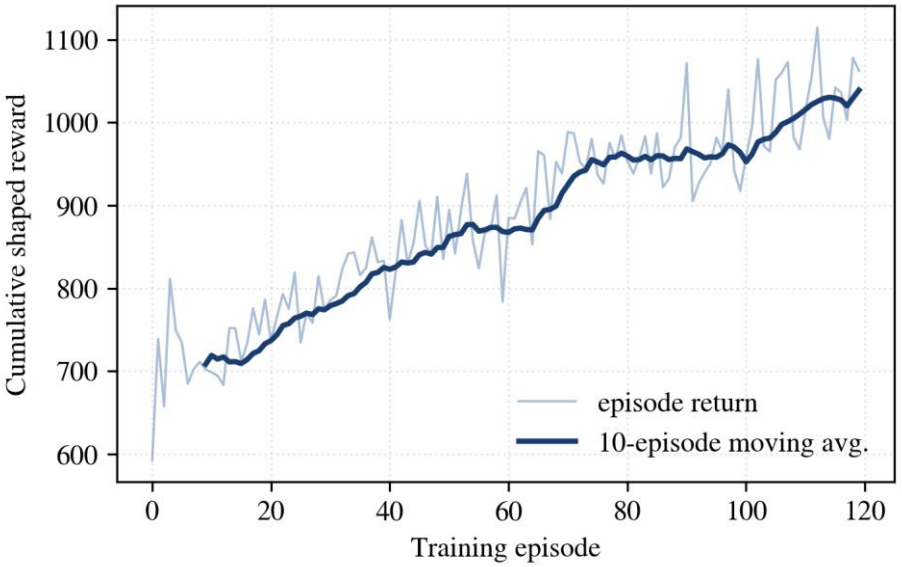

Fig. 2. Training return per episode (light) and 10-episode moving average (dark). The full 120-episode schedule takes ~2 minutes of wall-clock training on a 2-core VM; returns are still improving slowly at the end, partly because exploration has not fully decayed.

6.2 Energy and the Consolidation Trade-off

The headline comparison is shown in Table 2 and Fig. 3 . The EASQ with 23.43±0.77 kWh/day is better than the spread by 27.7%, round-robin by 27.0%, least-loaded by 24.3% and best-fit by 5.7% . Against every baseline the paired wilcoxon statistic is W=0 (EASQ wins on all five traces); with n=5 the two-sided p-value floors at 0.0625, so we treat significance as indicated but not established - a larger seed count is needed for a stronger statistical claim, and we prefer stating that to inflating n with correlated traces.

Table 2. Main comparison, mean ± std over five held-out traces. Bold marks the best value per row. No scheduler rejected any container.

Metric (per day)	Round-Robin	Spread	Least-Loaded	Best-Fit	EASQ
Energy (kWh)	32.08 ± 0.34	32.42 ± 0.30	30.93 ± 0.26	24.86 ± 0.54	**23.43 ± 0.77**
Active-node util.	0.36 ± 0.01	0.37 ± 0.01	0.30 ± 0.01	**0.60 ± 0.02**	0.39 ± 0.02
Stranded cap. (%)	70.1 ± 0.6	70.4 ± 0.6	70.3 ± 0.7	**21.4 ± 1.6**	40.3 ± 1.2
Over-load samples	17.4 ± 9.3	29.4 ± 5.4	**0.0 ± 0.0**	41.0 ± 12.5	**0.0 ± 0.0**
Util. im-balance	0.16 ± 0.01	0.18 ± 0.01	**0.06 ± 0.00**	0.30 ± 0.01	0.23 ± 0.01

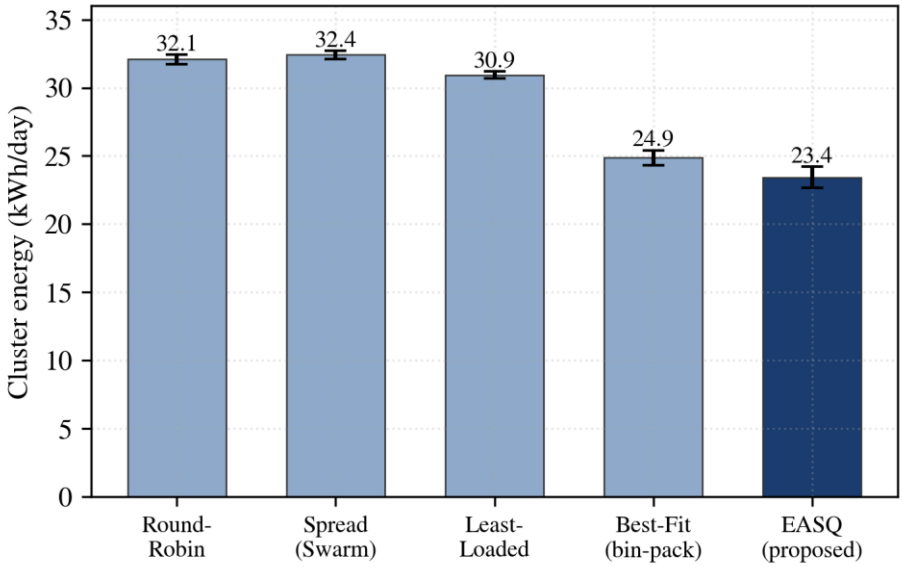

Fig. 3. Daily cluster energy across five held-out traces (error bars: ±1 std). EASQ's margin over best-fit (5.7%) is far smaller than over the spreading policies (~27%), and the honest comparison is against best-fit.

Where does EASQ's energy edge over best-fit come from, given that best-fit packs *harder* (60% active-node utilisation vs. 39%)? The timeline in Fig. 5 shows the mechanism. Best-fit packs by requested allocation, so nodes fill to their request ceiling while true usage lags, and during the overnight trough its request-based scoring keeps mid-sized nodes awake that EASQ lets drain and sleep. EASQ also learned a preference we

did not design in: during ramp-up it lands the 2,000 m class on already-active large nodes (where the marginal watt per millicore is lowest) rather than waking small ones. This surprised us; nothing in the state distinguishes node classes, but rank correlates with class under diurnal load, and the agent exploits the correlation.

The obvious objection is that the comparison is unfair to best-fit on safety and unfair to EASQ on waste, and both directions deserve numbers. Best-fit's packing buys 21.4% stranded capacity against EASQ's 40.3%—the clearest result *against* our method in this study, and worse than we anticipated—but pays 41 overload samples per day, each a throttling episode a latency SLO would feel. EASQ recorded zero overloads on all five traces while still beating best-fit's energy. Least-loaded also achieves zero overloads, but at 30.93 kWh/day: safety by keeping everything powered and empty. The three policies thus occupy distinct corners of an energy–waste–risk triangle (Fig. 4), and EASQ's claim is not dominance in all corners but the best energy-at-zero-overload point we observed.

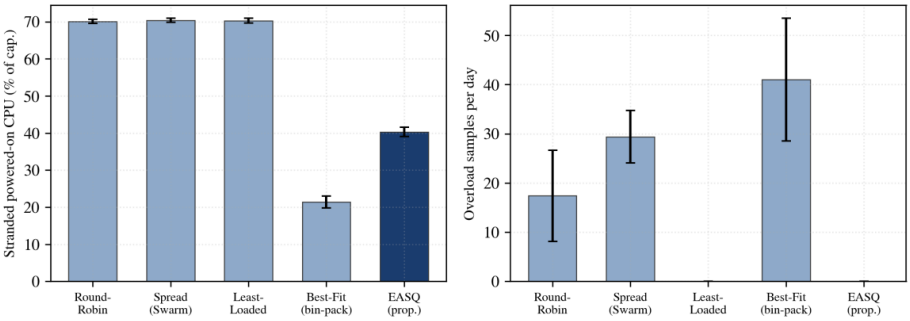

Fig. 4. The two faces of consolidation. Left: best-fit strands the least powered-on capacity; EASQ halves the spreading policies' waste but does not match bin packing. Right: the price—best-fit incurs 41 overload samples/day; EASQ and least-loaded incur none.

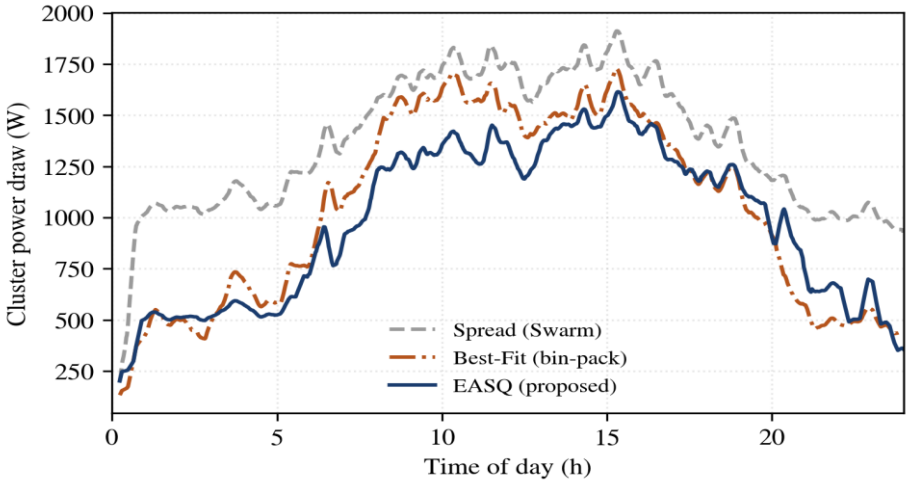

Fig. 5. Cluster power over one evaluation day (15-min smoothing, seed 0). EASQ tracks best-fit through the daytime peak but drops below it in the trough, where learned consolidation lets more nodes sleep; the spreading policy never falls under ~1.1 kW.

6.3 Ablation: What Each Reward Term Buys

Retraining with individual reward terms removed (Fig. 6) decomposes the design. Without the packing term ($w_u = 0$) energy climbs to 27.38 ± 0.52 kWh/day ($+3.95$): the pure energy signal is too sparse to drive consolidation on its own, since most placements on active nodes cost nearly the same small ΔP. Without the energy term ($w_e = 0$) the loss is smaller than we expected—24.65 ± 0.61, $+1.22$ kWh—because packing below θ already approximates consolidation; the marginal-power charge mainly teaches *which* node to wake when waking is unavoidable. With no overload penalty (w_o=0) energy is statistically indistinguishable from the full reward (24.23±0.68) but overloads jump from zero to 28.0±11.8 per day, the term exists purely to buy safety and it costs about 0.8 kWh/day to have it.

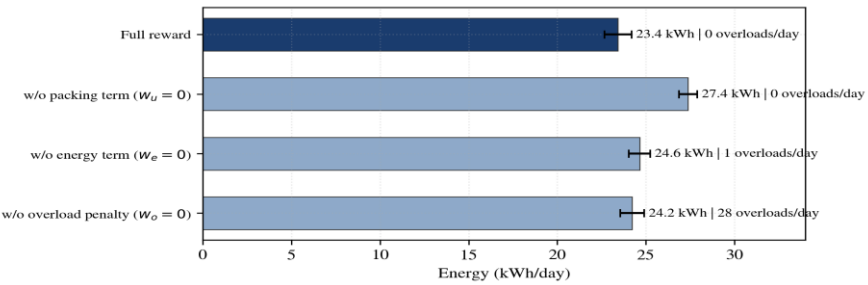

Fig. 6. Reward ablation (five seeds each; error bars ± 1 std). Each variant retrains from scratch for 120 episodes. Annotations give energy and overload counts; the overload penalty changes safety, not energy.

6.4 Sensitivity

Fig. 7 sweeps the two learning hyper-parameters. The learning rate has a wide tolerance: α∈[0.05,0.3] stays in 24.9–22.6 kWh/day, with α=0.3 actually outperforming our default. But it collapses at α=0.5 (28.51±0.52), where single noisy updates overwrite cell estimates faster than averaging repairs them. The discount sweep is more directly embarrassing for our default: $γ=0.90$ reaches $21.45±0.37\,kWh/day$, 8.4% below the $γ=0.95$ configuration reported everyplace else in this paper, while $γ=0.99$ degrades sharply to $27.59±0.72$. Mechanically this is coherent, the shaped reward already internalizes each action's consequence, so long bootstrapping horizons mostly propagate trace noise, but we fixed γ=0.95 before running the sweep and report the better setting as a sensitivity finding rather than silently promoting it to the headline. Had we tuned on the evaluation seeds, the abstract would say 21.5 kWh. The lack of this is deliberate."

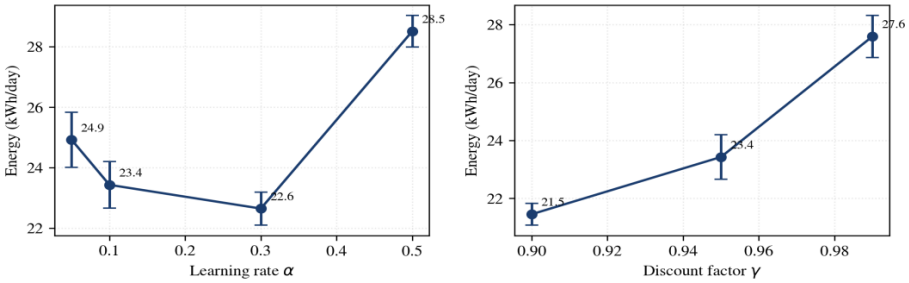

Fig. 7. Sensitivity to hyperparameters (five seeds per point). Left: energy vs. learning rate alpha. Performance drops off steeply only at alpha=0.5. Right: shorter discount horizons help; $\gamma=0.90$ beats the default $\gamma=0.95$ used in all other experiments.

7 Threats to Validity and Limitations

Simulation is the largest one. The linear power model (Eq. 3) omits frequency scaling, thermal effects, and the 10–30 s latency and energy cost of actual sleep/wake transitions; on hardware, frequent node parking could claw back part of the 1.4 kWh/day margin over best-fit, and only a testbed study will settle by how much. Synthetic workloads are the second: our diurnal Poisson process with Beta-distributed usage matches aggregate statistics of published traces [1] but not their burst structure or inter-container correlation. Third, the stranded-capacity gap (40.3% vs. best-fit's 21.4%) means EASQ's zero-overload operation is partly purchased with headroom; a datacentre billing on capacity rather than energy would weigh this differently. Fourth, five evaluation seeds cap the achievable Wilcoxon p at 0.0625 and leave effect-size estimates coarse. Finally, the rank-based action space cannot separate equally-utilised nodes with different efficiency, EASQ handles placement only (no migration or vertical scaling), and

the 80-state encoding was chosen by intuition—whether a learned discretisation or a small critic network preserves auditability while closing the waste gap is open. Scalability to clusters larger or more heterogeneous than the eight nodes studied here (Sect. 4.1) is likewise untested and, given the M-dependent action space, would require retraining rather than direct reuse of the learned table.

8 Conclusion and Future Work

This paper set out to test whether a deliberately small reinforcement learner can make container placement energy-aware without sacrificing the auditability operators require, and the simulation evidence says yes with qualifications. A 640-entry Q-table trained in about two minutes cut daily energy 27.7% against Swarm-style spreading and 5.7% against Kubernetes-style bin packing while eliminating the overload events that packing by requests incurs; the marginal-power reward proved to be the load-bearing design choice, its delayed cluster-level predecessor having failed by a 6.9 kWh/day margin. The method's weaknesses are equally concrete: it strands nearly twice the powered-on capacity of best-fit, its statistical support is bounded by five seeds, and its default discount factor was beaten by its own sensitivity sweep.

We stress that every number above is a simulation result: real sleep/wake latency, DVFS and thermal effects, measurement noise, and correlated production workloads are not represented in the simulator, so these findings should be read as evidence of a promising mechanism rather than a demonstrated energy saving on physical hardware.

Next steps follow from those weaknesses in order: a physical Kubernetes deployment via a scheduler-extender with RAPL-based power measurement, workload replay from public production traces, more evaluation seeds for proper effect sizes, and an action-space extension that exposes node efficiency classes. The doctoral research this paper belongs to will pursue exactly that sequence.

Acknowledgments. The authors thank the Department of Computer Science and Engineering, Banasthali Vidyapith, for computational support. Simulator source code, all trace seeds, and scripts regenerating every table and figure are available from the corresponding author.

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

References

1. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proc. 3rd ACM Symposium on Cloud Computing (SoCC), pp. 1–13. ACM (2012)
2. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., Wilkes, J.: Large-scale cluster management at Google with Borg. In: Proc. 10th European Conference on Computer Systems (EuroSys), pp. 1–17. ACM (2015)
3. Barroso, L.A., Hölzle, U.: The case for energy-proportional computing. Computer **40**(12), 33–37 (2007)

4. Fan, X., Weber, W.-D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. In: Proc. 34th International Symposium on Computer Architecture (ISCA), pp. 13–23. ACM (2007)
5. Marathe, N., Gandhi, A., Shah, J.M.: Docker Swarm and Kubernetes in cloud computing environment. In: Proc. 3rd International Conference on Trends in Electronics and Informatics (ICOEI), pp. 179–184. IEEE (2019)
6. Meena, J.K., Banyal, R.K.: Efficient virtualization in cloud computing. In: Proc. 5th International Conference on Computing Methodologies and Communication (ICCMC), pp. 227–232. IEEE (2021)
7. The Kubernetes Authors: Kubernetes documentation: scheduling, eviction, and resource management. https://kubernetes.io/docs/, last accessed 2026/08/22 (2024)
8. Kavitha, B., Varalakshmi, P.: Performance analysis of virtual machines and Docker containers. In: Data Science Analytics and Applications (DaSAA 2017), pp. 99–113. Springer (2018)
9. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. 2nd edn. MIT Press, Cambridge (2018)
10. Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource management with deep reinforcement learning. In: Proc. 15th ACM Workshop on Hot Topics in Networks (HotNets), pp. 50–56. ACM (2016)
11. Hernandez, M., Patel, D.: Dynamic resource allocation using reinforcement learning in cloud environments. Journal of Cloud Computing: Advances, Systems and Applications **13**(1), 45–59 (2024)
12. Zhang, Y., Liu, B., Gong, Y., Huang, J., Xu, J., Wan, W.: Application of machine learning optimization in cloud computing resource scheduling and management. In: Proc. 5th International Conference on Computer Information and Big Data Applications, pp. 171–175 (2024)
13. Sandhu, R., Faiz, M., Kaur, H., Srivastava, A., Narayan, V.: Enhancement in performance of cloud computing task scheduling using optimization strategies. Cluster Computing **27**, 1–24 (2024)
14. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
15. Saxena, V., Kumar, A.: A reinforcement learning-inspired, energy-efficient CPU scheduling approach for container orchestration in cloud environments. In: Proc. Workshop on Advanced Computing and Intelligent Information Systems (WACIIS). (2026, in press)
16. Acharya, J., Suthar, A.C.: Container scheduling algorithm in Docker based cloud. Webology **19**(2) (2022)
17. Barik, R.K., Lenka, R.K., Rao, K.R., Ghose, D.: Performance analysis of virtual machines and containers in cloud computing. In: Proc. International Conference on Computing, Communication and Automation (ICCCA), pp. 1204–1210. IEEE (2016)
18. Potdar, A.M., Narayan, D.G., Kengond, S., Mulla, M.M.: Performance evaluation of Docker container and virtual machine. Procedia Computer Science **171**, 1419–1428 (2020)
19. Elbelgehy, A.G.A., AbdElrazek, S., El-Bakry, H.M., Riad, A.M.: Performance evaluation of virtual cloud labs using hypervisor and container. Indian Journal of Science and Technology **13**(48), 4646–4653 (2020)
20. Lwin, J.: Enhancing cloud task scheduling with multi-objective optimization using K-means clustering and dynamic resource allocation. Journal of Computing Theories and Applications **2**(2), 202–211 (2024)
21. Nawrocki, P., Smendowski, M.: A survey of cloud resource consumption optimization methods. Journal of Grid Computing **23**(1), 5 (2025)

22. Almeida, F., Silva, T.: Cost-efficient load balancing algorithms for cloud computing. Future Generation Computer Systems **135**, 200–213 (2025)
23. Jang, J., Jung, J., Hong, J.: An efficient virtual CPU scheduling in cloud computing. Soft Computing **24**(8), 5987–5997 (2020)
24. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency and Computation: Practice and Experience **24**(13), 1397–1420 (2012)
25. Schwartz, R., Dodge, J., Smith, N.A., Etzioni, O.: Green AI. Communications of the ACM **63**(12), 54–63 (2020)
26. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP. In: Proc. 57th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 3645–3650 (2019)
27. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8**(3–4), 279–292 (1992)
28. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience **41**(1), 23–50 (2011)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (http://creativecommons.org/licenses/by-nc/4.0/), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

