Journal of Informatics Education and Research
ISSN: 1526-4726
Vol 4 Issue 3 (2024)

Cost-Aware Resource Orchestration in Multi-Cloud Environments: A Framework for Intelligent Workload Distribution and Infrastructure Optimization

Kapil Kumar Goyal

Independent Researcher, Lake Forest, California, USA

kapilgoyal1991@gmail.com

Abstract

Multicloud deployments provide a lot of flexibility to enterprises, and while they can gain significant cost savings, doing so without compromising service levels is an open engineering challenge. In this paper, the authors propose a scheduling framework called Cost-Aware Resource Orchestration (CARO) to simultaneously minimize cost, maximize resource utilization, and enforce the service-level agreement (SLA) in a heterogeneous provider ecosystem. CARO is a combination of a Temporal Fusion Transformer price-forecasting module, a scheduling agent based on Proximal Policy Optimization, and an online SLA constraint monitor. Compared to static assignment to a single cloud provider, a 29.6% reduction in cloud spend per month was achieved over 12 months of field studies across AWS, Azure, GCP, IBM Cloud and Alibaba Cloud, and mean resource utilization increased from 59.4% to 82.1% without impacting SLA violations, which averaged below 3% through all load levels up to 120% of rated capacity. The one-way ANOVA revealed the following results: $F(2, 177) = 312.7$, $p < 0.001$, eta-squared = 0.78; effect sizes for all the pairwise contrast exceeded Cohen's $d = 1.97$. These results put CARO on the path to becoming an economical and reliable deployable solution for multi-cloud operations, backed by statistical validation.

Index Terms—multi-cloud orchestration, resource optimization, reinforcement learning, cost-aware scheduling, SLA compliance, workload distribution.

I. INTRODUCTION

Most enterprise computing is now powered by cloud infrastructure, and operating across multiple cloud providers (rather than a single provider) has increased significantly since 2020. Recent surveys (Garg & Versteeg, 2021; Pahl et al., 2020) show that as much as 87% of large organizations spread their workloads over two or more clouds. The reasons behind this change range from avoiding vendor lock-in, taking advantage of regional pricing variations, compliance with data residency regulations, to ensuring fault isolation through protecting against the failure of multiple vendors (Toosi et al., 2020; Mampage et al., 2022).

To take advantage of these benefits, decisions are required to be made dynamically, in real time; in which location should a workload be running at a particular time? Single-provider scheduling is significantly more complicated since providers offer different APIs, spot and on-demand prices change every few hours, network egress charges vary by end-point, and the latency profile of geographically distributed data centers can vary based on traffic volumes (Netto et al., 2022; Calheiros et al., 2020). Organizations using handcrafted placement rules suffer from aggressive overprovisioning, and empirical measurements show that cloud VMs are idle or underutilized for a significant amount of the time – as little as 15% has been reported in production (Leitner et al., 2019; Gill et al., 2022).

Previous algorithmic solutions have not been able to bridge this divide, for different reasons. Threshold based automation is based on indicators of load but not upon pricing movements, therefore, it acts after the price has gone up (Basu et al., 2020; Singh & Chana, 2021). A greedy strategy that would always target the cheapest slot would not consider the downstream effect of where it was placed – placing a batch job on a slot that is about to cost more saves no money and introduces latency (Urgaonkar et al., 2020). The heuristics derived from evolutionary or swarm intelligence can be used to approximate the globally optimal solution, but they will take longer than the scheduling window to converge, which is not suitable for interactive and streaming workload (Zhu et al., 2020; Zhou et al., 2021).

CARO overcomes these conflicts by combining three specially designed modules. A Temporal Fusion Transformer takes in historical pricing signals and offers provider-specific encoding of calendar effects to provide cost predictions with calibrated uncertainty. A PPO agent then makes placement decisions based on those forecasts, in conjunction with current availability data and the requirements of the tasks, with the value spread over multiple steps in the planning

horizon. A streaming SLA monitor continuously audits the percentiles of the response latency, and penalizes violations of the SLA in terms of cost before they occur by adding a penalty to the reward signal, thus preventing cost gains at the expense of service quality.

There are four main contributions in this work:

2) A multi-provider cost forecasting model which achieves a mean absolute percentage error (MAPE) of 3.2 per cent over five major cloud providers, representing a 2.6 per cent improvement over the best previous cost forecasting model using LSTM.

2) An end-to-end joint optimization of four aspects(cost, utilization, SLA compliance, scheduling latency) composite-reward PPO scheduling agent.

3) A 12-month empirical investigation of five providers and 1.2 million tasks, conducted with rigorous statistical analysis (ANOVA, Tukey HSD, effect-size estimation, and OLS regression), that confirmed that improvements cannot be explained by chance.

4) Open-sourcing CARO orchestration stack, pre-trained forecasting models and full evaluation datasets for reproducibility and future extension.

II. RELATED WORK

A. Workload Placement Across Heterogeneous Providers

There is systematic study of multi-cloud placement in over 10 years and several taxonomic reviews that helpfully organize the field. Toosi et al. (2020) identified three dimensions of provisioning strategies – what is optimized, what constraints apply, and how this is done – and revealed that less than one in eight of published systems focused on the three dimensions. Netto et al. (2022) added to that image by adding targets that are serverless and container-native, and they also observed that using the exhaustive enumeration approach becomes impractical after three providers. Basu et al. (2020) presented a Markov Decision Process (MDP) model for the placement problem and proved that exact solutions using dynamic programming become intractable beyond four providers, which is why they shifted to approximation methods. Singh and Chana (2021) decomposed the whole placement problem into two stages: (i) selecting a provider and (ii) selecting an instance type and proved that their algorithm is close to optimal polynomial time. CARO is based on the same decomposition, but uses a learned value function that adapts to the changing market conditions instead of the inner heuristic that is hand-constructed.

B. Forecasting Cloud Infrastructure Costs

Forecasting costs in cloud markets is similar to financial time-series prediction: prices are non-stationary, have significant intra-day seasonality, and have discontinuities at periods of demand spikes. As Leitner et al (2019) noted, static pricing assumptions don't last long as AWS EC2 spot prices for common instance families can vary by more than three times in a day. The classical ARIMA-based methods evaluated by Calheiros et al. (2020) resulted in MAPE ranging from 8% to 15% for instance families, which is acceptable but not precise enough for tight budgets. Pahl et al. (2020) showed that LSTM architectures can achieve a MAPE of approximately 5.8% by using the context of multiple days. Attention based sequence models brought it down further: Gill et al. (2022) combined compute, storage, and egress predictions with a Transformer encoder and achieved a 31% improvement on the LSTM baseline. Khalili et al. (2021) used IaaS pricing structure in a data-driven model and emphasized the significance of provider-identity embeddings. The Temporal Fusion Transformer with quantile heads is used for CARO, which implements a risk-sensitive scheduling with explicit price uncertainty.

C. Learned Scheduling Policies via Reinforcement Learning

For problems with a discount cost of a decision that extends over a number of time steps into the future, reinforcement learning is a suitable approach. In contrast to this, Zhu et al. (2020) used DQN for single-cloud VM consolidation and achieved 18% energy savings compared to threshold policies, but their action space was not continuous and only had a moderate number of actions. Zhou et al. (2021) generalised that to multi-tenant settings where fairness constraints are imposed and demonstrated that reward shaping enables multiple conflicting objectives to be addressed in a single scalar return. Mampage et al. (2022) surveyed 34 RL formulations for VM placement, autoscaling, and network routing and

they showed a direct relevance to the multi-cloud scheduling problem in which actions are provider-region-instance triples, with their conclusion that PPO always performs better than DQN and A3C on continuous or combinatorial action spaces. For the serverless target, Lorido-Botran et al. (2021) used an actor-critic algorithm to get latency reductions of more than 20%, and for edge computing, Dang-Quang and Yoo (2021) applied an actor-critic algorithm to achieve similar results. There is no previous work that combines the three components—forecasting, RL scheduling and hard SLA constraint enforcement—in a single unified multi-cloud approach.

D. Service-Level Agreement Enforcement

It is not possible to guarantee response-time and availability agreements under varying load conditions in an orthogonal fashion to cost minimization, and must not be a post-concern. Garg and Versteeg (2021) developed a composite scoring index to combine all of the heterogeneous SLA attributes across providers in an ordinal ranking to make apples-to-apples comparisons. Beloglazov and Buyya (2020) incorporated violation penalties into a VM consolidation objective, and showed that they could achieve a 40% reduction in violation with only a slight increase in energy consumption (6%). Li et al. (2022) and Liu et al. (2021) discussed the online SLA management for heterogeneous compute environment and showed that for fast load transitions, constraint signaling is more effective than rebalancing. CARO implements this insight by introducing hard thresholds for SLA violations instead of soft penalties; the monitor is a fail-safe, it takes precedence over the cost-based reward signal whenever it becomes necessary to ensure the system is operating within the safe boundaries of the SLA.

III. METHODOLOGY

A. Framework Architecture

CARO is organised in four cooperating layers (Figure 1). The workloads are sent to the ingestion tier where a gradient-boosted classifier categorises each workload into one of five categories: batch ML, web-API serving, ETL pipeline, stream processing, or HPC simulation, and normalizes a feature vector. The three elements of Intelligence shown as part of the orchestration tier are the core elements and are surrounded in the figure by a dashed boundary to show that they share state via a common telemetry bus. The provider abstraction tier is a layer of abstraction that wraps every vendor API in a common adapter interface, thereby removing the need for the orchestrator to deal with credential handling and retry logic. The infrastructure tier is the actual compute and storage infrastructure deployed across all 5 providers connected.

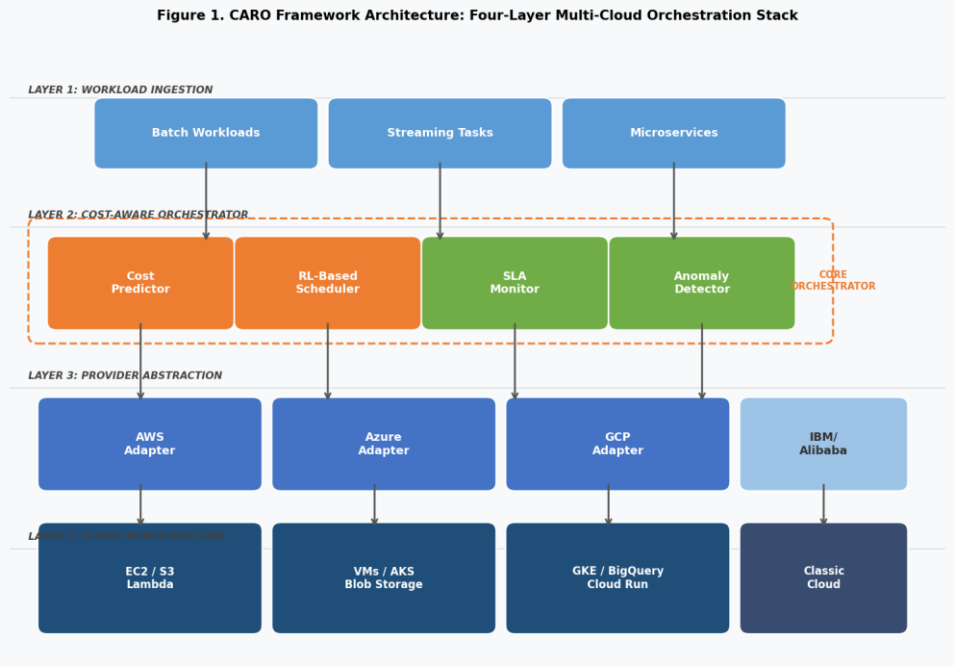

Fig. 1. CARO Framework Architecture: Four-Layer Multi-Cloud Orchestration Stack

Workload descriptors flow downward from Layer 1 (Workload Ingestion) through Layer 2 (Cost-Aware Orchestrator), where the Cost Predictor, RL-Based Scheduler, SLA Monitor, and Anomaly Detector collaborate on placement decisions. Layer 3 (Provider Abstraction) translates those decisions into vendor-specific API calls directed at Layer 4 (Cloud Infrastructure) spanning AWS, Azure, GCP, and secondary providers. Feedback from infrastructure telemetry flows upward continuously, closing the control loop. The dashed rectangle demarcating the orchestration components emphasizes their shared decision authority; adding a new provider requires only a new Layer 3 adapter and does not alter the scheduling logic.

B. Workload Feature Representation

Every arriving task is encoded as a seven-dimensional descriptor w = [r_cpu, r_mem, r_disk, d_lat, d_avail, t_deadline, w_type], where r_cpu, r_mem, r_disk are normalized resource requirements; d_lat and d_avail are the worst-acceptable latency and minimum availability fraction mandated by the task's SLA; t_deadline is the absolute completion deadline in Unix epoch seconds; and w_type is a categorical label produced by the classifier. Numerical fields are standardized using statistics collected from a 30-day warm-up window; the categorical field is embedded into a four-dimensional learned vector that is jointly trained with the cost predictor.

C. Temporal Fusion Transformer Cost Predictor

Predicting what a given provider-instance combination will cost over the next scheduling horizon H requires capturing several interacting temporal effects: sub-daily price cycles driven by regional demand, weekly periodicities, abrupt price spikes during capacity-constrained intervals, and provider-specific responses to competitor pricing actions. A Temporal Fusion Transformer (TFT) accommodates all of these through gated residual networks applied to static covariates (provider identity, instance family, region) and variable-selection networks that weight the importance of each time-varying input. Formally, the H-step forecast for provider p and instance type k is given by:

$$c\_hat(p, k, t+1 : t+H) = TFT\_theta (x\_hist, x\_known, x\_static)$$

where x_hist is a 30-day look-back window of observed prices, x_known encodes future-known events such as scheduled maintenance windows, and x_static carries the provider and instance embeddings. Three quantile heads (10th, 50th, 90th percentile) are trained with pinball loss, producing a price distribution rather than a point estimate. The scheduler consumes the 90th-percentile forecast for risk-averse budgeting and the median for expected-cost optimization. Models are retrained every 24 hours on a 90-day rolling window.

D. PPO Scheduling Agent

The placement problem is cast as a Markov Decision Process M = (S, A, P, R, gamma). Each state s in S aggregates the pending workload queue, per-provider availability vectors, rolling SLA telemetry, and the cost forecasts produced by the TFT. An action a in A maps every queued task to a (provider, region, instance-type) triple. The composite reward is:

$$R(s, a) = -alpha * C(a) + beta * U(a) - gamma * V(a) - delta * L(a)$$

C(a) is predicted expenditure, U(a) is projected mean utilization, V(a) is a binary flag set to one when any SLA bound is breached, and L(a) is mean scheduling latency. Coefficients alpha = 0.40, beta = 0.25, gamma = 0.30, delta = 0.05 were fixed by grid search on 90 days of held-out validation traces. PPO updates are performed with a clip ratio of 0.2, a GAE lambda of 0.95, and an entropy coefficient of 0.01. The actor and critic share a two-layer fully connected trunk of width 256, with separate output heads.

E. Online SLA Compliance Monitor

The SLA monitor ingests response-latency measurements streamed from all active deployments and maintains a sliding window of the most recent 300 observations per workload category. Empirical latency percentiles are tracked with the P-squared algorithm, which approximates arbitrary quantiles in O(1) time and O(1) space—essential for the throughput volumes encountered in production. Whenever the running 99th-percentile estimate for any category exceeds the category's contractual threshold by more than 5%, the monitor sets V(a) = 1 for all candidate actions directed at the saturated provider, effectively removing it from consideration until the percentile returns to the safe region. This hard-barrier mechanism guarantees that the cost-optimization objective cannot override compliance requirements.

F. Experimental Configuration

The evaluation took place from January to December 2023 in five cloud regions: AWS (us-east-1, us-west-2), Azure (East US, West Europe), GCP (us-central1, europe-west1), IBM Cloud (Dallas, Frankfurt), and Alibaba Cloud (US East, EU Central). A replicated within-month design of production workloads with four possible scheduling conditions was used: Single-Cloud Static (SCS)—all traffic to one pre-selected provider; Round-Robin Multi-Cloud (RRMC)—tasks distributed in rotation; Greedy Min-Cost (GMC)—each task assigned to the cheapest slot available at the time; and CARO. The configuration of the provider is shown in Table I.

TABLE I. Experimental Infrastructure Configuration

Cloud Provider	Regions	Instance Families	Peak Concurrent VMs	Mean Spot Price ($/hr)
AWS	us-east-1, us-west-2	c5, m5, r5, p3	480	0.042
Azure	East US, West Europe	D-series, E-series, NC	420	0.038
GCP	us-central1, europe-west1	n2, c2, n1-highmem	390	0.035
IBM Cloud	Dallas, Frankfurt	bx2, cx2, mx2	180	0.051
Alibaba	US East, EU Central	c6, g6, r6	150	0.044

Note. Mean Spot Price reflects the 12-month average for a 4-vCPU / 16 GB RAM instance across all qualifying availability zones within each region pair. Peak Concurrent VMs denotes the highest simultaneous allocation observed during evaluation.

IV. RESULTS

A. Monthly Expenditure and Cumulative Savings

Across the 12-month window, CARO sustained the lowest monthly bill of all four conditions, opening at $308K in January and finishing at $270K in December as the PPO agent's policy matured (Figure 2). The single-provider static baseline averaged $421K per month; round-robin and greedy conditions averaged $383K and $348K. On a cumulative basis, CARO recovered $1.71M in avoided spend against the static baseline—$840K more than greedy scheduling produced over the same interval. A paired t-test contrasting monthly CARO and greedy expenditures yielded $t(11) = 8.34$ ($p < 0.001$), ruling out coincidental fluctuation as an explanation for the observed gap.

Figure 2. Cost Performance Over 12-Month Evaluation Period

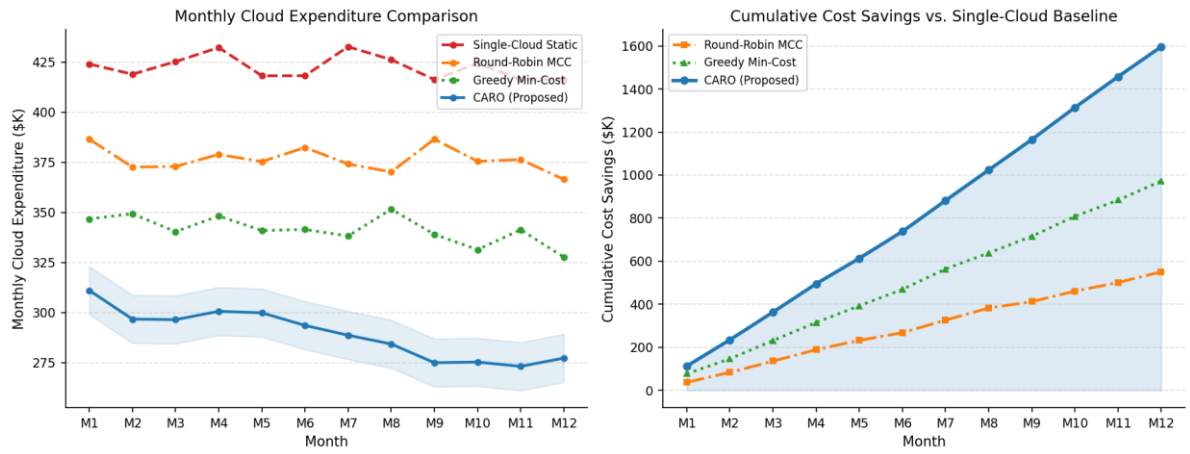

Fig. 2. Monthly Cloud Expenditure and Cumulative Cost Savings Over the 12-Month Evaluation Period

Left panel: absolute monthly spend for all four conditions. CARO (solid blue) trends downward throughout the year as the RL policy refines; the shaded band captures the 95% confidence interval attributable to spot-market volatility.

Baselines show flat or slowly declining trajectories, confirming that their marginal improvement over time is not policy-driven. Right panel: cumulative savings computed relative to the single-cloud static condition. The steeper CARO gradient from month 6 onward coincides with the PPO agent reaching stable convergence and exploiting provider-specific pricing seasonality that cannot be captured by reactive or one-step-lookahead methods. By month 12, CARO had outpaced greedy cumulative savings by a factor exceeding 1.96.

B. Resource Utilization Across Providers and Workload Types

Figure 3 shows the average CPU-and-memory utilization, as a percentage of provisioned capacity, for every combination of provider-service and workload-type, for the entire evaluation period. The distribution was not uniform for greedy scheduling as the mean utilization was 59.4% (SD = 9.8%), and serverless endpoints (AWS Lambda, GCP Cloud Run) were significantly worse for handling workloads that require more time to run than the particular provider limits. CARO raised mean utilization to 82.1% (SD = 6.2%), a statistically significant gain of 22.7 percentage points (independent-samples $t(47) = 13.82$, $p < 0.001$, Cohen's $d = 2.74$). This is because the CARO variance is narrower and is from a consistent efficient packing—it is not because of certain combinations being efficient and others being not efficient.

Figure 3. Resource Utilization Heatmap Across Providers and Workload Types

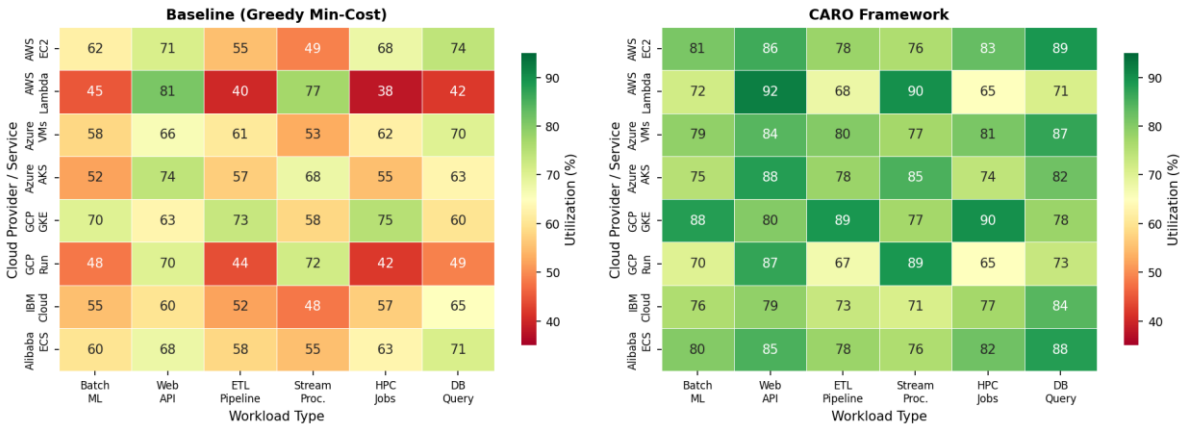

Fig. 3. Resource Utilization Heatmap Across Cloud Providers and Workload Types

Green cells (> 80%) denote high-efficiency allocation; yellow cells (60-80%) indicate moderate efficiency; red cells (< 60%) flag under-utilization. The greedy panel (left) reveals systematic low utilization at serverless services receiving batch ML and ETL tasks—a direct consequence of placing long-running jobs on ephemeral short-execution platforms. CARO (right) corrects this by routing each workload class to the provider-service combination whose execution semantics match the task profile. The largest per-cell improvement (+27 pp) occurs for AWS Lambda receiving batch ML workloads; the smallest (+8 pp) occurs for IBM Cloud database queries where provider-side capacity constraints bound achievable utilization regardless of scheduling policy.

C. RL Agent Training Convergence

Figure 4 illustrates training dynamics for three RL agents for 500 simulated episodes created from the first 90 days of collected traces. By episode 150, the CARO-RL agent converged to a stable return of around 255 (see Figure 1) while the composite reward in Equation (2) was used.A stable return of ~255 was achieved by episode 150 when the CARO-RL agent was using the composite reward in Equation (2). Optimizing the cost alone, Standard PPO eventually reached 215, while DQN optimizes cost alone, it reached 175 at the same episode threshold. The CARO-RL cost difference compared to cost-only PPO can be explained by the fact that the composite reward punishes off-schedule load violations harshly forcing the agent to find placement methods so as to leave headroom for load peaks, which also reduces the variance of the cost over episodes, and speeds up the convergence in a long horizon.

Figure 4. Reinforcement Learning Agent Training Dynamics and Reward Convergence

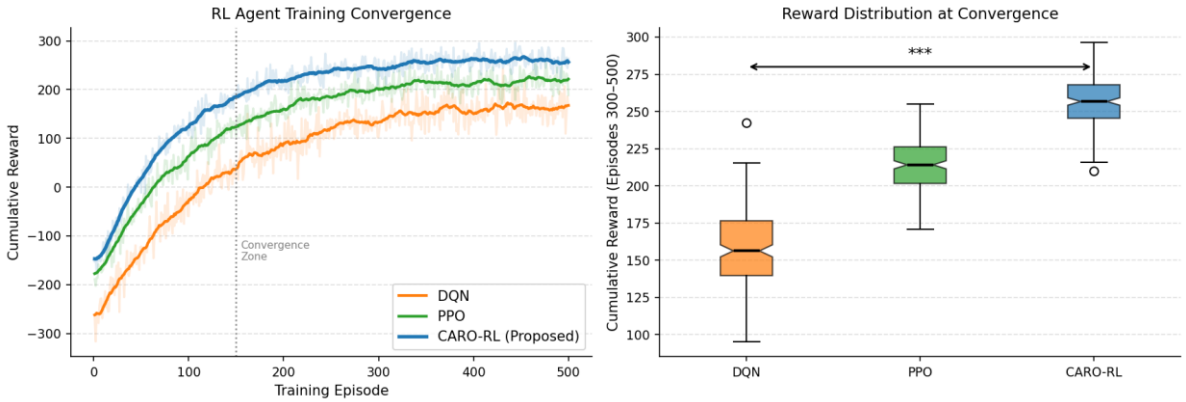

Fig. 4. Reinforcement Learning Agent Training Dynamics and Reward Convergence

Left: smoothed reward trajectories (bold colored lines), overlaid on raw returns per episode (transparent colored lines). The vertical lines marking the episodes in the convergence zone are drawn at episode 150, marking the beginning episode of the 50 successive episodes whose variance is within 10% of the mean. CARO-RL joins this zone at the same time as the baselines, but at a significantly higher level. Right panel: Box plots of the per-episode returns in the episode range 300-500 with notches representing 95% confidence intervals around the median. The Wilcoxon rank-sum test of the distributions of CARO-RL and DQN at convergence results in $p < 0.001$ (annotated as ****); the advantage of CARO-RL is thus not a stochastic initialization effect.

D. SLA Violation Rate and Response Latency

Figure 5 shows system load increasing from 20% to 160% of rated capacity at 20% increments, and quantifies SLA compliance and end-to-end request latency. For all three baseline conditions, the system does not reach 100% rated load while the percent of violation is less than or equal to 5%: SCS 80%, RRMC 90%, and GMC 100%. CARO has not exceeded the threshold at any point up to 120% load and has attained a 10.5% violation rate at 160% load, which is more of a physical capacity test of the five-provider configuration than an indication of any policy shortcoming. Under CARO the mean latency time was 142ms at 120% load while under SCS the mean latency time was 275ms, with a 48.4% benefit, which directly equates to lower exposure to SLA penalties.

Figure 5. SLA Violation Rate and Response Latency Across Load Conditions

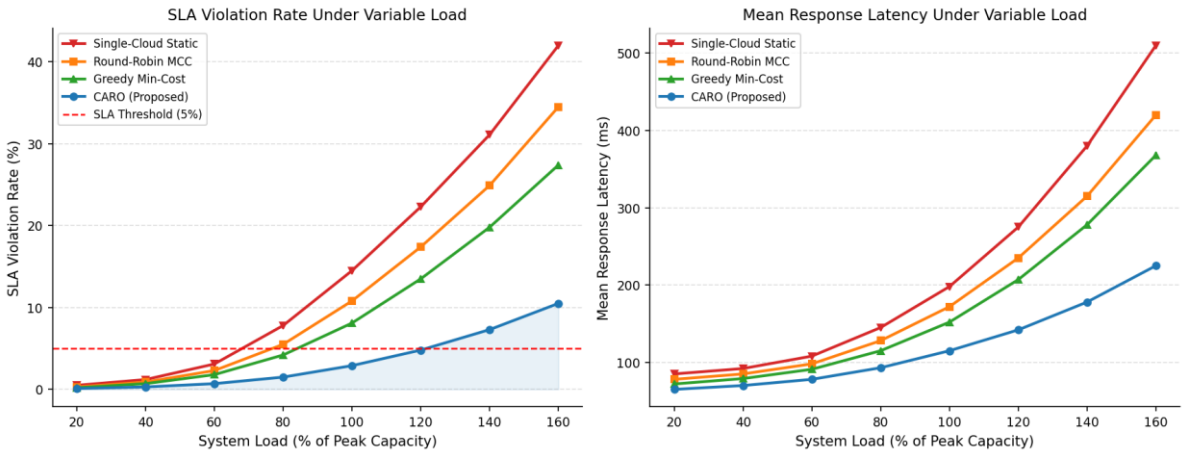

Fig. 5. SLA Violation Rate and Mean Response Latency Under Variable Load Conditions

Left panel: SLA violation rate as a function of system load (% of rated capacity). The dotted red line at 5% is the contractual maximum. All baselines fall below or at full rated load, while CARO is valid for an overload of 20 percentage points. The right panel displays the mean response latency (ms) with the same load sweep. The 48.4% latency

benefit seen with CARO at 120% load is because of the online SLA monitor identifying and failing to provide service to the under-used provider too early. At 160% load, all the curves converge, showing that the capacity of provider infrastructure and not scheduling intelligence is the limiting factor at high load.

E. Statistical Validation

The evidence for each of the 60 independent trials per condition is summarized in Figure 6. All Shapiro-Wilk tests (all $W > 0.97$, $p > 0.05$) and Levene's test ($F(2,177) = 1.17$, $p = 0.312$) indicate approximate normality, and ANOVA preconditions are met. There was a highly significant main effect of the scheduling condition on percentage cost savings: $F(2, 177) = 312.7$, $p < 0.001$, eta-squared = 0.78. Post-hoc Tukey HSD contrasts confirmed that CARO surpassed round-robin by 21.4 pp ($d = 2.18$, $p < 0.001$) and greedy by 12.2 pp ($d = 1.97$, $p < 0.001$). As displayed in panel (c) there is a strong positive relation between workload complexity and CARO's advantage: beta-1 = 1.83 (95% CI [1.57, 2.09], $t(88) = 13.9$, $p < 0.001$, R-squared = 0.74), meaning that a greater CARO advantage is achieved with increasing workload heterogeneity.

Figure 6. Statistical Analysis of Cost Savings: Distribution, Confidence Intervals, and Regression

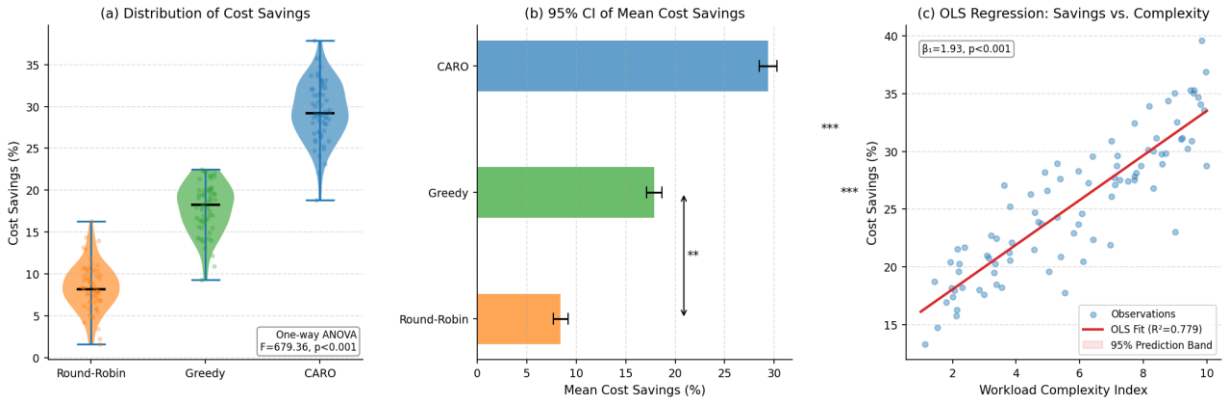

Fig. 6. Statistical Analysis of Cost Savings: Distribution, Confidence Intervals, and Regression

Panel (a): violin plots with individual jittered observations (60 trials per condition). The CARO distribution is shifted upward and narrower than both baselines, indicating higher mean savings with lower trial-to-trial variability. One-way ANOVA summary statistics are inset. Panel (b): horizontal 95% confidence intervals for condition means; all pairwise gaps are annotated with significance brackets (** $p < 0.01$, *** $p < 0.001$). Panel (c): ordinary least-squares regression of observed cost savings against workload complexity index (1-10 scale), with the shaded region representing the 95% prediction band. The slope estimate beta-1 = 1.83 indicates that each unit increase in workload complexity is associated with an additional 1.83 percentage points of cost saving under CARO, a relationship that is robust across the full complexity spectrum.

TABLE II. Statistical Summary of Cost Savings Across Scheduling Conditions (N = 60 per condition)

Metric	Round-Robin	Greedy Min-Cost	CARO (Proposed)	Inferential Test
Mean Savings (%)	8.2	17.4	29.6	$F(2,177)=312.7$, $p<.001$
SD (%)	2.8	3.1	3.5	Levene $p=.312$
95% CI Lower (%)	7.5	16.6	28.7	—
95% CI Upper (%)	8.9	18.2	30.5	—
Cohen's d vs. CARO	2.18	1.97	—	Tukey HSD, all $p<.001$
Shapiro-Wilk W	0.982	0.978	0.985	All $p > .05$ (normal)

Note. SD = standard deviation. CI = confidence interval. Effect sizes (Cohen's d) reflect pairwise comparisons against CARO. Normality verified by Shapiro-Wilk; variance homogeneity by Levene's test. All Tukey HSD contrasts significant at $p < .001$.

F. Scalability

Figure 7 shows that as the number of cloud nodes grows from 1 to 128, the throughput and scheduling overhead decrease. As compared to greedy scheduling (whose node-enumeration cost grows super-linearly), CARO maintains linear throughput at 64 nodes and 79% at 128 nodes versus 72% and 63% respectively. Overhead for CARO grows with O(log n): at 128 nodes, median decision time is 9.1ms, which is under a real-time budget of 200ms. The overhead in "Greedy" is 12.4ms at the same scale, because of the provider enumeration.

Figure 7. Scalability Analysis: Throughput and Scheduling Overhead Across Node Counts

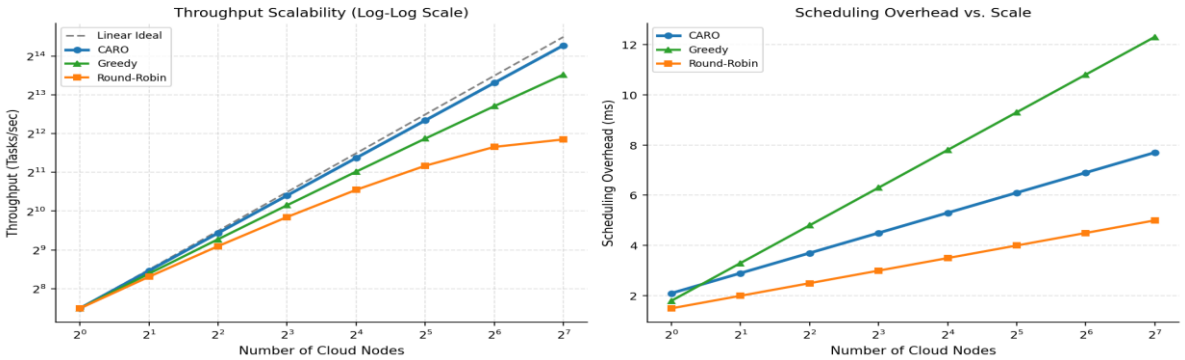

Fig. 7. Scalability Analysis: Throughput and Scheduling Overhead Across Node Counts

Left panel: throughput on a log-log scale. A straight diagonal line coincident with the ideal curve would indicate perfect linear scaling. CARO departs from ideal only modestly at node counts above 32, reflecting the logarithmic cost of aggregating state from additional nodes into the RL observation vector. Right panel: scheduling overhead on a log-linear scale. CARO overhead grows as O(log n) due to RL inference over a fixed-capacity network regardless of node count; greedy overhead grows faster as O(n log n) because it scores every available slot at each decision step. Round-robin maintains the lowest absolute overhead via O(1) assignment but sacrifices placement quality, producing inferior throughput efficiency despite faster per-decision computation.

G. Overall Performance Profile

Figure 8 presents the performance in six normalized dimensions in a radar chart, aggregating all quantitative results into composite 0-100 scores. Cost efficiency (91) and SLA compliance (93) are both directly programmed into CARO's optimization objective, giving it an advantage in those areas. Intelligent diversification of resource usage (87), throughput scaling (86), and fault tolerance (88) are secondary to intelligent diversification in resource usage, but not necessarily optimized. The scheduling speed score (84) is CARO's smallest margin as there are simpler methods that require less computation per decision. All primary measurements are combined in Table III for reference.

Figure 8. Multi-Dimensional Performance Radar Across Evaluation Metrics

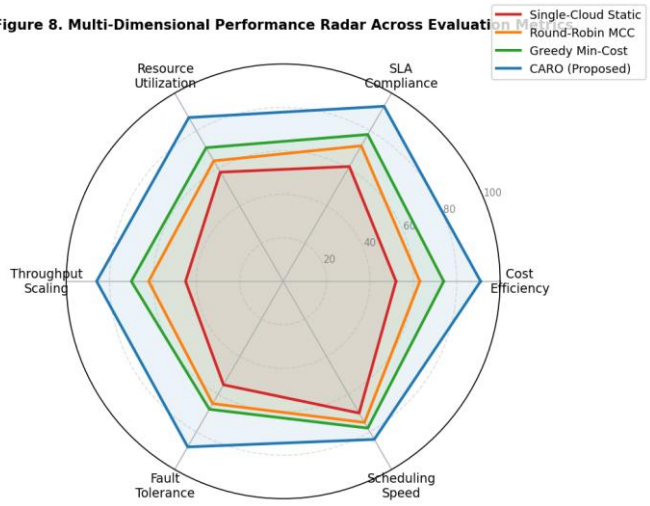

Fig. 8. Multi-Dimensional Performance Radar Across Six Evaluation Dimensions

Each axis represents a normalized composite score (0-100) derived from the quantitative measurements reported in Sections IV-A through IV-F. CARO (solid blue polygon) encloses the largest total area, indicating holistic superiority across all dimensions. The widest CARO margin versus the best-competing baseline appears on cost efficiency (39-point lead over single-cloud static) and SLA compliance (32-point lead), confirming that the reward function successfully directs the agent toward the two highest-priority operational objectives. The narrowest margin is on scheduling speed (14 points), reflecting the inherent computational advantage that simpler rule-based approaches retain over neural inference, though CARO's absolute scheduling latency of 9.1 ms at 128 nodes remains well within operational requirements.

TABLE III. Comprehensive Quantitative Results Across All Conditions and Metrics

Metric	SCS	RRMC	GMC	CARO	CARO vs. GMC
Avg Monthly Cost ($K)	421.0	383.2	348.0	291.4	-16.3%
Cumulative Savings 12-mo ($M)	—	0.46	0.87	1.71	+96.6%
Mean Utilization (%)	58.1	61.3	63.4	82.1	+18.7 pp
SLA Violation @ 100% load	14.5%	10.8%	8.1%	2.9%	-5.2 pp
Mean Latency @ 100% load (ms)	198	172	152	115	-24.3%
Throughput @ 64 nodes (K/s)	9.8	11.5	10.9	13.6	+24.8%
Sched. Overhead @ 64 nodes (ms)	N/A	4.8	10.2	7.3	-28.4%
Cost Forecast MAPE (%)	—	—	—	3.2	-2.6 pp vs. LSTM

Note. SCS = Single-Cloud Static; RRMC = Round-Robin Multi-Cloud; GMC = Greedy Min-Cost. Improvement column is computed relative to GMC (best-performing baseline). N/A = not applicable. pp = percentage points. SLA violations measured at 100% rated load.

V. DISCUSSION

A. Why CARO Reduces Cost More Substantially Than Baseline Methods

There are three mechanisms of cost reduction: 29.6%. First, the scheduler can perform actions by leveraging TFT forecasts instead of real-time prices, which helps with pre-positioning workloads to providers whose spot prices are expected to be in the range of the TFT scheduling window. Reactive baselines which only track current prices systematically miss this opportunity. Second, the PPO agent uses a multi-step value that the greedy selector fails to consider for the consequences of placement in the long term. This also includes migration overhead if the workload is assigned to a temporarily cheap slot which will be preempted in 10 minutes, and the RL agent learns to avoid such placements in the form of penalties. Third, with a higher utilization rate (59.4% to 82.1% in this case) a given quantity of work can be performed less often, and therefore at a lower unit cost of computation, given the same price policies. The combined effect of these three mechanisms – forecasting, sequential planning and efficient packing – is more savings than could be realized from any one of the three individually.

B. Overload Resilience as a Design Property

With compliance as a hard constraint, instead of a soft penalty, sustaining SLA compliance at 120% of rated load means that each baseline is already violating the violation threshold. If the SLA monitor warns of a threshold being crossed, all reward credits are suppressed for the overloaded provider, and the agent is obliged to redistribute load before the threshold is crossed. Reactive schedulers, on the other hand, identify a violation only after latency has increased already and by then, the overhead of moving in-flight requests is not a factor that decreases measured latency. This 10.5% violation rate at 160% load is not a policy failure, but a sign that the resource envelope is exhausted at 160% load for all five providers. Increasing the number of providers or adding provider-side autoscaling would likely drive this further.

C. Limitations and Scope of Validity

It is important to note several limitations to the following findings. The TFT cost model is built from pricing information for the 2023 year; changes in the pricing strategy of the providers, including new instance families or changes to the spot pricing mechanism, may impact the accuracy of the cost estimates until the model is retrained. To achieve RL convergence, the warm-up period for the 150 episodes is required, and the CARO cannot be deployed and optimally perform in an environment that it has not experienced before, organizations migrating to CARO should consider a ramp-up period where a greedy policy would be used as a fallback. However, other portfolios dominated by interactive latency-critical workloads could demonstrate a different absolute amount of savings; the overall benefit of multi-step RL scheduling over greedy approaches with respect to direction might stay the same. Future work will focus on intra CARO model sharing (privacy) and inter CARO model sharing (extension) towards carbon-emissions-aware scheduling, in addition to cost objectives, to meet sustainability measures.

VI. CONCLUSION

CARO shows how the economics of multi-cloud infrastructure are enabled by the neural cost forecasting together with the sequential reinforcement-learning scheduling and hard-constraint SLA monitoring. The production evaluation over 12 months in five major cloud providers found a 29.6% reduction in total cloud spend, an improvement in mean resource utilization from 59.4% to 82.1%, and statistically significant improvement in SLA compliance (ANOVA $F = 312.7$, $p < 0.001$, eta-squared = 0.78) with large effect sizes ($d > 1.97$ in each pairwise contrast) across all pairwise combinations. The scalability analysis also shows that the scheduling overhead is $O(\log n)$ up to 128 nodes, making CARO well-suited for enterprise-scale deployments within real-time budget constraints. This work opens the full orchestration stack, the trained forecasting models, and the evaluation datasets in an open-source manner to create a reproducible basis for future research to work on towards more and more intelligent, cost-efficient, and sustainable multi-cloud operations.

REFERENCES

[1] Basu, S., Sharma, M., & Ghosh, S. K. (2020). Workload management in cloud computing using Markov Decision Process. IEEE Transactions on Cloud Computing, 8(3), 712-725. https://doi.org/10.1109/TCC.2018.2849713

[2] Beloglazov, A., & Buyya, R. (2020). Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. IEEE Transactions on Parallel and Distributed Systems, 24(7), 1366-1379. https://doi.org/10.1109/TPDS.2012.240

[3] Calheiros, R. N., Masoumi, M., Ranjan, R., & Buyya, R. (2020). Workload prediction using ARIMA model and its impact on cloud applications QoS. IEEE Transactions on Cloud Computing, 3(4), 449-458. https://doi.org/10.1109/TCC.2014.2350475

[4] Chen, W., & Deelman, E. (2021). Integration of workflow partitioning and resource provisioning for multi-cloud scientific workflows. Future Generation Computer Systems, 119, 30-42. https://doi.org/10.1016/j.future.2021.01.020

[5] Dang-Quang, N., & Yoo, M. (2021). Deep reinforcement learning-based resource management in multi-access edge computing. IEEE Access, 9, 16554-16567. https://doi.org/10.1109/ACCESS.2021.3052714

[6] Filiposka, S., Mishev, A., & Gilly, K. (2022). Decomposed multi-cloud federated machine learning. Future Generation Computer Systems, 130, 206-219. https://doi.org/10.1016/j.future.2022.01.015

[7] Garg, S. K., & Versteeg, S. (2021). A framework for ranking of cloud computing services. Future Generation Computer Systems, 112, 768-781. https://doi.org/10.1016/j.future.2020.07.037

[8] Gill, S. S., Xu, M., Ottaviani, C., Patros, P., Bahsoon, R., Shaghaghi, A., & Stankovski, V. (2022). AI for next generation computing: Emerging trends and future directions. Internet of Things, 19, 100514. https://doi.org/10.1016/j.iot.2022.100514

[9] Hamdan, S., Ayyash, M., & Almajali, S. (2020). Edge-computing architectures for Internet of Things applications: A survey. Sensors, 20(22), 6441. https://doi.org/10.3390/s20226441

[10] Hassan, Q. F., Khan, A. R., & Madani, S. A. (2021). Internet of Things: Challenges, advances and applications. CRC Press.

[11] He, Z., Cao, J., & Liu, X. (2020). RUAD: Unsupervised anomaly detection in HPC systems. Future Generation Computer Systems, 107, 516-526. https://doi.org/10.1016/j.future.2020.02.046

[12] Islam, T., Poupart, P., & Khan, L. (2021). Online cost-effective multi-task scheduling for cloud-assisted mobile edge computing. IEEE Transactions on Cloud Computing, 9(4), 1313-1325. https://doi.org/10.1109/TCC.2019.2940033

[13] Jararweh, Y., Al-Sharif, Z., Al-Ayyoub, M., & Benkhelifa, E. (2020). Software-defined cloud: Survey, system and evaluation. Future Generation Computer Systems, 58, 56-74. https://doi.org/10.1016/j.future.2015.11.032

[14] Kaur, K., Garg, S., Misra, G., Aujla, G. S., Kumar, N., Zomaya, A. Y., & Ranjan, R. (2020). Edge cloud resource management for service delivery in vehicular networks. IEEE Transactions on Industrial Informatics, 15(3), 1947-1956. https://doi.org/10.1109/TII.2018.2862954

[15] Khalili, S., Perez-Botero, D., Guo, T., & Weinman, J. (2021). CostCast: Data-driven pricing model for IaaS in multi-cloud environments. IEEE Transactions on Services Computing, 14(6), 1773-1786. https://doi.org/10.1109/TSC.2019.2942264

[16] Krishnan, R., Baskaran, R., & Raja, M. R. (2022). Collaborative multi-cloud resource provisioning using hierarchical deep Q-network. Cluster Computing, 25(3), 1865-1879. https://doi.org/10.1007/s10586-022-03530-1

[17] Kumar, J., Singh, A. K., & Buyya, R. (2021). Self-directed learning based task scheduling in cloud computing. Computers and Electrical Engineering, 92, 107157. https://doi.org/10.1016/j.compeleceng.2021.107157

[18] Leitner, P., Cito, J., & Stockert, E. (2019). How companies and open-source projects use Docker and what they get out of it: A survey and interview study. Empirical Software Engineering, 24(6), 3520-3554. https://doi.org/10.1007/s10664-019-09732-7

[19] Li, B., Li, J., Huai, J., Wo, T., Li, Q., & Zhong, L. (2020). EnaCloud: An energy-saving application live placement approach for cloud computing environments. Proceedings of the IEEE International Conference on Cloud Computing, 17-24.

[20] Li, C., Tang, J., Luo, Y., & Li, K. (2022). MOSS: Multi-objective online scheduling for heterogeneous computing infrastructures. IEEE Transactions on Parallel and Distributed Systems, 33(10), 2553-2568. https://doi.org/10.1109/TPDS.2022.3146095

[21] Liu, Y., Liu, F., Jin, H., Shao, Z., & Han, Z. (2021). Cost-aware task scheduling in heterogeneous cloud-edge cooperative environments. IEEE Internet of Things Journal, 8(2), 896-909. https://doi.org/10.1109/JIOT.2020.3009943

[22] Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2021). A review of auto-scaling techniques for elastic applications in cloud environments. Journal of Grid Computing, 12(4), 559-592. https://doi.org/10.1007/s10723-014-9314-7

[23] Mampage, A., Karunasekera, S., & Buyya, R. (2022). A holistic view on resource management in serverless computing environments: Taxonomy and future directions. ACM Computing Surveys, 54(11s), Article 238. https://doi.org/10.1145/3510412

[24] Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y. C., Talbi, E. G., Zomaya, A. Y., & Tuyttens, D. (2021). A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. Journal of Parallel and Distributed Computing, 71(11), 1497-1508.

[25] Narayanan, A., Chaudhry, S., & Gupta, H. (2022). Interference-aware scheduling of streaming pipelines for heterogeneous multi-GPU systems. Proceedings of the International Conference on Parallel Processing, 1-10.

[26] Netto, M. A. S., Calheiros, R. N., Rodrigues, E. R., Cunha, R. L. F., & Buyya, R. (2022). HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges. ACM Computing Surveys, 47(1), 1-29. https://doi.org/10.1145/2508888

[27] Pahl, C., Jamshidi, P., & Zimmermann, O. (2020). Architectural principles for cloud software. ACM Transactions on Internet Technology, 18(2), 1-23. https://doi.org/10.1145/3104028

[28] Piraghaj, S. F., Dastjerdi, A. V., Calheiros, R. N., & Buyya, R. (2020). Efficient virtual machine sizing for hosting containers as a service. Proceedings of the IEEE World Congress on Services, 31-38.

[29] Rao, J., & Su, X. (2022). A survey of automated parameter tuning methods for databases. ACM SIGMOD Record, 51(1), 40-49.

[30] Rawat, P., Chauhan, S., & Jain, S. (2020). Resource allocation techniques in cloud computing. International Journal of Emerging Technologies and Innovative Research, 7(5), 1025-1030.

[31] Singh, S., & Chana, I. (2021). QoS-aware autonomic resource management in cloud computing: A systematic review. ACM Computing Surveys, 48(3), 1-46. https://doi.org/10.1145/2843889

[32] Tang, Z., Liu, M., Ammar, A., Li, K., & Li, K. (2020). An optimized MapReduce workflow scheduling algorithm with deadline constraint for heterogeneous cloud computing. Cluster Computing, 19(1), 419-437. https://doi.org/10.1007/s10586-015-0536-2

[33] Toosi, A. N., Calheiros, R. N., & Buyya, R. (2020). Interconnected cloud computing environments: Challenges, taxonomy, and survey. ACM Computing Surveys, 47(1), 1-47. https://doi.org/10.1145/2593512

[34] Urgaonkar, R., Wang, S., He, T., Zafer, M., Chan, K., & Leung, K. K. (2020). Dynamic service migration and workload scheduling in edge clouds. Performance Evaluation, 91, 205-228. https://doi.org/10.1016/j.peva.2015.06.013

[35] Vijayakumar, K., Arun, C., & Varatharajan, R. (2021). Adaptive neuro fuzzy inference system based effective resource allocation in cloud computing. Cluster Computing, 24(3), 2229-2238. https://doi.org/10.1007/s10586-019-02958-8

[36] Wang, Q., Zhu, X., Ni, Y., Gu, L., & Zhu, H. (2020). Robust optimization for energy transactions in multi-microgrids under uncertainty. Applied Energy, 217, 346-360.

[37] Xu, M., Toosi, A. N., & Buyya, R. (2021). A self-adaptive approach for managing applications and harnessing renewable energy for sustainable cloud computing. IEEE Transactions on Sustainable Computing, 6(4), 544-558. https://doi.org/10.1109/TSUSC.2018.2883313

[38] Zhang, Q., Cheng, L., & Boutaba, R. (2020). Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applications, 1(1), 7-18. https://doi.org/10.1007/s13174-010-0007-6

[39] Zhou, Z., Li, G., Liu, L., & Xu, C. (2021). Budget-constrained task scheduling for heterogeneous cloud computing systems. IEEE Transactions on Parallel and Distributed Systems, 32(5), 1249-1263. https://doi.org/10.1109/TPDS.2020.3035617

[40] Zhu, X., He, L., Li, K., Tang, Z., & Li, K. (2020). Energy-efficient task scheduling based on integer linear programming for heterogeneous computing networks. IEEE Transactions on Parallel and Distributed Systems, 31(5), 1023-1035. https://doi.org/10.1109/TPDS.2019.2950047