

AI-Enabled Cloud Resource Management for Improving Quality of Service and Reducing Computational Cost

Dhirendra Kumar¹, Gautam Kumar², Anand Kamal^{3,*}

¹Assistant Professor, Department of Computer Science and Engineering, Government Engineering College Madhubani. Email: profdhiru@gmail.com

²Assistant Professor, Department of Computer Science and Engineering, Government Engineering College Madhubani. Email: gtm10117@gmail.com

³Assistant Professor, Department of Computer Science and Engineering, Government Engineering College Madhubani. Email: anandkamal2003@gmail.com

* Corresponding author: Anand Kamal (anandkamal2003@gmail.com)

Abstract: Cloud data centres must honour strict Quality of Service (QoS) commitments while the cost of leased and powered computing capacity keeps rising. Rule-based elasticity mechanisms react only after demand has already changed, forcing operators to choose between wasteful over-provisioning and SLA-breaching under-provisioning. This paper presents an AI-enabled resource management framework organised as a monitor–forecast–decide–act loop. A gradient-boosted regression model predicts the request rate one provisioning delay ahead, and a cost-aware controller converts the forecast into a virtual-machine (VM) target through a validation-derived safety margin, an online residual correction and hysteresis on scale-in. The framework is evaluated in a discrete-time simulator on ten independent synthetic workloads containing diurnal, weekly and flash-crowd components, and is compared with static peak provisioning, a reactive threshold auto-scaler and a linear-regression predictive scaler. The proposed controller lowered daily cost by 15.9% relative to the reactive policy and by 54.8% relative to static provisioning, matched the reactive SLA-violation rate (1.05% versus 1.04%) and reduced 95th-percentile latency by 16.8%. The study also documents where the approach falls short, notably during bursts that exceed the range seen in training.

Keywords: Cloud computing; resource management; auto-scaling; workload forecasting; gradient boosting; quality of service; cost optimisation

1. Introduction

Cloud computing has become the default delivery model for enterprise software, analytics and consumer services because it offers computing capacity as a metered, on-demand utility [1, 2]. Elasticity is realised only when capacity follows demand, yet request rates follow daily and weekly cycles overlaid with sudden flash crowds, and newly requested virtual machines (VMs) need several minutes before they can serve traffic. Operators must therefore keep enough capacity online to meet the latency commitments in service-level agreements (SLAs) while avoiding payment for machines that sit idle. The energy side of this trade-off is significant; data centres were estimated to account for roughly one percent of global electricity use in 2018, and their consumption depends increasingly on how well capacity is matched to load [3].

The dominant industrial practice is still rule-based scaling, in which capacity is added when a monitored metric such as CPU utilisation crosses an upper threshold and removed when it falls below a lower one. Such rules are simple to operate but inherently backward-looking: by the time a threshold is breached the surge has begun, and the provisioning delay guarantees a window of degraded service. Surveys of auto-scaling research [4, 5] identify proactive techniques, which anticipate load instead of chasing it, as a way out of this dilemma. Large operators have meanwhile



shown that learning from historical usage can raise cluster efficiency in production [6, 7], and recent reviews argue that artificial intelligence (AI) will be central to the next generation of cloud management [8, 9].

Despite this progress, a gap remains between forecasting research and operational resource management. Many studies report predictor accuracy through measures such as the mean absolute percentage error (MAPE) but do not show how that accuracy translates into SLA compliance and cost once the forecast is turned into scaling actions. A forecast is not a decision: because forecasts are uncertain and under-provisioning is penalised differently from over-provisioning, a controller must decide how far to trust them. This paper studies the complete path from prediction to provisioning under an explicit QoS constraint.

The main contributions of this work are as follows:

1. A formulation of cloud capacity management as cost minimisation under a response-time constraint that explicitly includes the VM provisioning delay.
2. A modular monitor–forecast–decide–act framework in which a gradient-boosted workload forecaster feeds a cost-aware controller equipped with a validation-derived safety margin, online residual correction and scale-in hysteresis.
3. A reproducible simulation study over ten independent workloads that compares the framework with static, reactive and linear-regression-based provisioning using cost, utilisation, latency and SLA metrics.
4. An analysis of where the approach helps and where it does not, including its behaviour during bursts that lie outside the range observed in training.

The remainder of the paper is organised as follows. Section 2 reviews related work, Section 3 formalises the problem, Section 4 describes the proposed framework, Section 5 explains the experimental design, Section 6 presents and discusses the results together with the limitations of the study, and Section 7 concludes.

2. Related Work

2.1 Rule-based and model-based auto-scaling

Auto-scaling techniques are commonly grouped into threshold rules, queueing and control-theoretic models, reinforcement learning and time-series analysis [4]. A later taxonomy extends this view to web applications and highlights the interplay between the scaling policy, the workload characteristics and the provisioning overhead [5]. Surveys of resource scheduling likewise emphasise that QoS, cost and energy should be optimised jointly rather than in isolation [10]. Early energy-aware heuristics showed that consolidating load onto fewer hosts can lower power consumption without violating SLAs when it is done with care [11].

2.2 Workload prediction

Forecasting demand is the natural first step towards proactive control. Calheiros et al. [12] showed that an ARIMA model can predict request arrivals accurately enough to support dynamic provisioning that meets QoS targets with efficient resource use. Deep models have since been applied to richer signals: a self-directed learning model was proposed for forecasting cloud workloads [13], and esDNN exploited multivariate inputs in a deep neural network for workload prediction [14]. Recurrent architectures such as long short-term memory networks [15] remain popular for sequence prediction, but they demand more data and tuning than tree ensembles. Gradient boosting [16] is a practical middle ground: it handles heterogeneous features, trains quickly and can be retrained often.

2.3 Learning-based decision making

Rather than separating prediction from control, reinforcement learning (RL) methods learn a provisioning or scheduling policy directly from interaction with the environment [17]. Mao et al. [18] demonstrated that a deep RL agent can learn to pack jobs with multi-resource demands and compete with hand-tuned heuristics. For edge–cloud

settings, actor–critic learning combined with residual recurrent networks has been used for dynamic scheduling under stochastic conditions [19], and machine-learning-based provisioning has been surveyed from the standpoint of reliability [20]. Learning has also been applied to neighbouring problems such as job-failure prediction [21]. RL policies, however, need long training, careful reward design and safe exploration, and are harder to audit than a forecast combined with an explicit rule.

2.4 Summary and research gap

Table 1 summarises the families of approaches discussed above. Comparatively few studies isolate how the design of the controller that consumes a forecast, in particular its treatment of forecast uncertainty and its scale-in behaviour, affects the QoS–cost trade-off. The present work concentrates on that link.

Table 1. Representative resource-management approaches and their main trade-offs.

Approach	Technique	Strength	Limitation
Threshold rules [4, 5]	Utilisation triggers	Simple and widely deployed	Reactive; lags behind demand
Statistical forecasting [12]	ARIMA time series	Interpretable and proactive	Limited for non-linear, bursty load
Deep forecasting [13, 14, 15]	Recurrent / deep networks	Captures non-linear patterns	Data- and tuning-intensive
Learned policies [18, 19]	Deep reinforcement learning	End-to-end decision learning	Training cost; limited transparency
This work	Gradient boosting + cost-aware controller	Explicit QoS–cost trade-off; lightweight	Tree models cannot extrapolate beyond training range

3. Problem Formulation

Time is divided into control intervals of length $\Delta = 5$ min indexed by t . Let λ_t denote the aggregate request rate in interval t , N_t the number of active identical VMs, and μ the request rate a single VM can serve at full utilisation. Utilisation is then

$$\rho_t = \lambda_t / (N_t \mu) \quad (1)$$

Using a common analytical approximation for a single-queue service, the mean response time is modelled as

$$R_t = \min\{s_0 t, R_{\text{cap}}\} \quad (2)$$

where s_0 is the service time on an unloaded VM and R_{cap} bounds the value when the system saturates. An SLA violation occurs in interval t when $R_t > R_{\text{max}}$. Solving Eq. (2) shows that this is equivalent to $\rho_t > \rho_{\text{max}} s_0 / R_{\text{max}}$; a 300 ms target with $s_0 = 60$ ms therefore gives $\rho_{\text{max}} = 0.8$.

A VM requested in interval t becomes usable only in interval $t + d$, where d is the provisioning delay, whereas releasing a VM takes effect immediately. If p is the price per VM-hour, the operating cost over T intervals is

$$C = \sum_{t=1}^T N_t \cdot p \cdot \Delta \quad (3)$$

and the management problem is to choose the provisioning sequence $\{N_t\}$ that solves

$$\min C + \beta \cdot \sum_t I[R_t > R_{\text{max}}] \quad \text{subject to } N_t \geq N_{\text{min}} \quad (4)$$

where $I[\cdot]$ is the indicator function, β weights SLA violations and N_{min} is a minimum pool size for availability. Because λ_{t+d} is unknown at the moment the decision that affects it must be taken, this is a problem of decision-making under uncertainty. That is what makes forecasting valuable and, at the same time, what makes blind trust in a forecast

dangerous: the cost of under-provisioning (SLA breaches and dropped requests) differs in kind from the cost of over-provisioning (wasted expenditure).

4. Proposed AI-Enabled Framework

4.1 Architecture

Figure 1 depicts the framework as a closed loop with five functional blocks. The monitoring layer collects request rate, utilisation and latency. The workload forecaster estimates demand at a horizon h equal to the provisioning delay. The AI decision engine combines that estimate with the QoS and cost constraints supplied by the QoS/cost model to compute a VM target, and the resource actuator applies it to the elastic VM pool. Realised metrics flow back to the monitoring layer, which lets the controller correct itself.

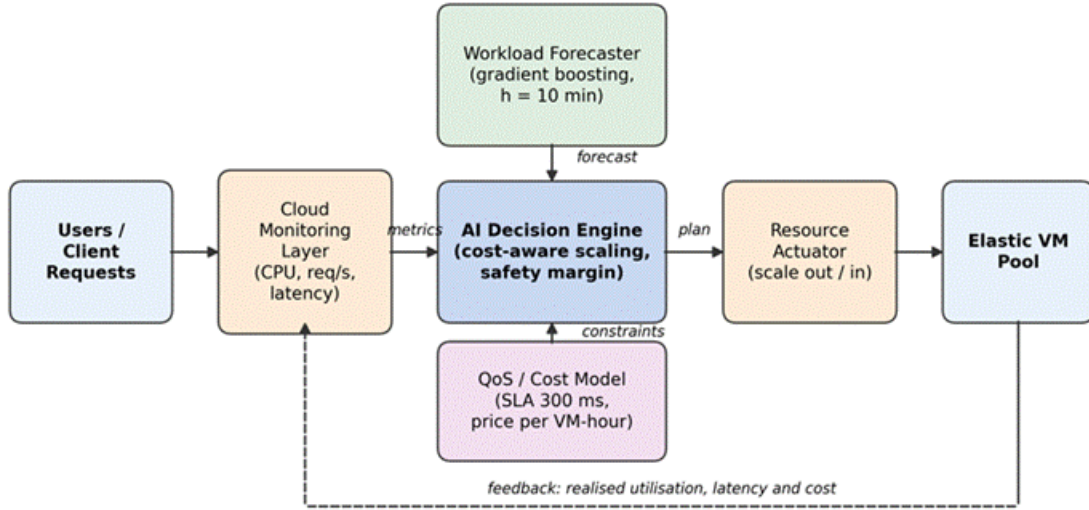


Fig. 1. Proposed monitor–forecast–decide–act control loop for AI-enabled cloud resource management.

4.2 Workload forecaster

The forecaster predicts λ_{t+h} from the information available at time t . The feature vector contains the current observation and the five preceding ones, the observation two hours earlier, the six-interval moving average, the demand recorded one day before the target time and in the interval preceding it, sine and cosine encodings of the time of day, a weekend indicator, and two short-term differences that capture the recent slope. A gradient boosting regressor [16] with 300 trees of depth three, a learning rate of 0.05 and a subsample ratio of 0.8 is trained on the first 14 days. These settings were fixed in advance and not tuned on the test period.

4.3 Cost-aware controller

The controller converts the point forecast $\hat{\lambda}_{t+h}$ into a VM target, with three safeguards against forecast uncertainty. First, a safety margin m is set to the 80th percentile of the positive relative forecast errors measured on a three-day validation window that follows the training period ($m = 5.2\%$ for the first workload). Second, an online correction term et , an exponentially weighted moving average (weight 0.2) of the relative under-prediction realised h intervals earlier, inflates the target when the forecaster has recently been too optimistic. Third, a reactive guard prevents the target from falling below the currently observed demand. The resulting target is

$$N^*_{t+h} = \max\{N_{\text{mint}+h}, 0.95 \lambda_t\} (1 + m + et) / (\mu u)$$

where u^* is a target utilisation below $p_{\max}$, so that residual forecast error is absorbed by the headroom between u^* and $p_{\max}$. The value $u^* = 0.65$ was selected from $\{0.60, 0.65, 0.70\}$ using the validation window only.

Scale-out orders are issued immediately for any shortfall between the target and the sum of active and in-flight VMs. Scale-in is deliberately slower: it occurs only when no order is pending and only after the target has stayed below 90% of the active pool for two consecutive intervals, in which case at most 15% of the pool is released. This hysteresis prevents oscillation when the forecast is noisy and reflects the asymmetry noted in Section 3, since releasing capacity too early is costlier than releasing it a few intervals late.

4.4 Control procedure

In every control interval the framework executes the following steps:

1. Observe λ_t and ρ_t , and update the correction term e_t using the forecast issued h intervals earlier.
2. Generate the forecast $\hat{\lambda}_{t+h}$ with the trained regressor.
3. Compute the VM target N^*_{t+h} with Eq. (5).
4. If the target exceeds the active plus pending VMs, request the difference; the new VMs become usable after d intervals.
5. Otherwise, if no order is pending and the target has been below 90% of the active pool for two consecutive intervals, release VMs.
6. Record cost and QoS metrics and return to step 1.

Each cycle involves a single model inference and constant-time arithmetic, so controller overhead is negligible compared with the five-minute control period.

5. Experimental Setup

5.1 Simulator and workloads

A discrete-time Python simulator implementing the model of Section 3 lets every policy face identical demand; evaluation on toolkits such as CloudSim [22] is left for future work. Each of ten random seeds generates an independent 21-day workload at five-minute resolution. Demand combines a base rate of 1,500 req/s, a 24-hour sinusoid with a 12-hour harmonic that peaks in the afternoon, a weekend factor of 0.78, multiplicative log-normal noise ($\sigma = 0.05$) and 42 flash crowds whose amplitude ranges from 35% to 100% above the underlying demand and whose duration, including ramp-up and decay, is roughly 45–90 min. The first 14 days train the forecaster, the following three days are used for validation and for choosing the safety margin and target utilisation, and the final four days (Thursday to Sunday) are used for evaluation. Table 2 lists the parameters.

Table 2. Simulation and controller parameters.

Parameter	Value	Remark
Control interval Δ	5 min	Monitoring and decision period
Per-VM capacity μ	50 req/s	At 100% utilisation
VM price p	US\$ 0.096 / h	Illustrative on-demand price
Provisioning delay $d = h$	2 intervals (10 min)	Scale-out only; scale-in immediate
Unloaded service time s_0	60 ms	Eq. (2)
SLA target $R_{\max}$	300 ms	Equivalent to $\rho_{\max} = 0.8$
Response-time cap R_{cap}	2 s	Applied at saturation

Minimum pool size N_{min}	2 VMs	Availability floor
Target utilisation u^*	0.65	Chosen on validation window
Workload length (train / val / test)	14 / 3 / 4 days	21 days per run
Independent runs	10	Different random seeds

5.2 Baselines and metrics

Three policies serve as baselines. *Static* provisioning sizes the pool so that the highest demand seen in training runs at 85% utilisation. *Reactive* scaling adds 25% capacity when utilisation exceeds 75% and releases 10% when it falls below 35%, subject to a 30-minute cooldown and no pending orders. *Predictive-LR* uses the same controller structure, target utilisation and scale-in hysteresis as the proposed method but replaces the forecaster with ordinary least squares and omits the safety margin, online correction and reactive guard, which isolates the contribution of those components. For forecast accuracy the naive persistence predictor and a six-interval moving average are also reported.

Performance is measured by daily cost, mean utilisation, the SLA-violation rate (percentage of intervals with $R_t > 300$ ms), mean and 95th-percentile latency, the share of requests exceeding capacity, and the number of scaling actions per day. Forecast quality is measured by MAE, RMSE and MAPE at horizon $h = 2$. All figures are means and standard deviations over the ten runs.

6. Results and Discussion

6.1 Forecast accuracy

Table 3 compares the predictors on the test days. The gradient-boosting forecaster attains the lowest error on every measure, with a MAPE of 6.56% against 7.31% for persistence (a 10.3% relative reduction) and 6.94% for linear regression. The margin is modest because persistence is a strong predictor at a ten-minute horizon, while the moving average lags rising demand. Figure 2 shows that both learned models track the diurnal pattern closely but that the boosted model under-predicts the largest flash crowd, since tree ensembles cannot output values beyond the range of their training targets. This is why the controller does not rely on the point forecast alone.

Table 3. Forecast accuracy at a 10-minute horizon on the test days (mean $\pm$ SD over ten runs; best value in bold).

Predictor	MAE (req/s)	RMSE (req/s)	MAPE (%)
Naive (persistence)	105.1 $\pm$ 6.8	183.6 $\pm$ 19.3	7.31 $\pm$ 0.33
Moving average (6 intervals)	130.2 $\pm$ 13.2	241.7 $\pm$ 34.3	9.43 $\pm$ 0.71
Linear regression	99.9 $\pm$ 5.8	173.9 $\pm$ 18.5	6.94 $\pm$ 0.29
Gradient boosting (proposed)	94.3 $\pm$ 7.6	171.2 $\pm$ 23.8	6.56 $\pm$ 0.47

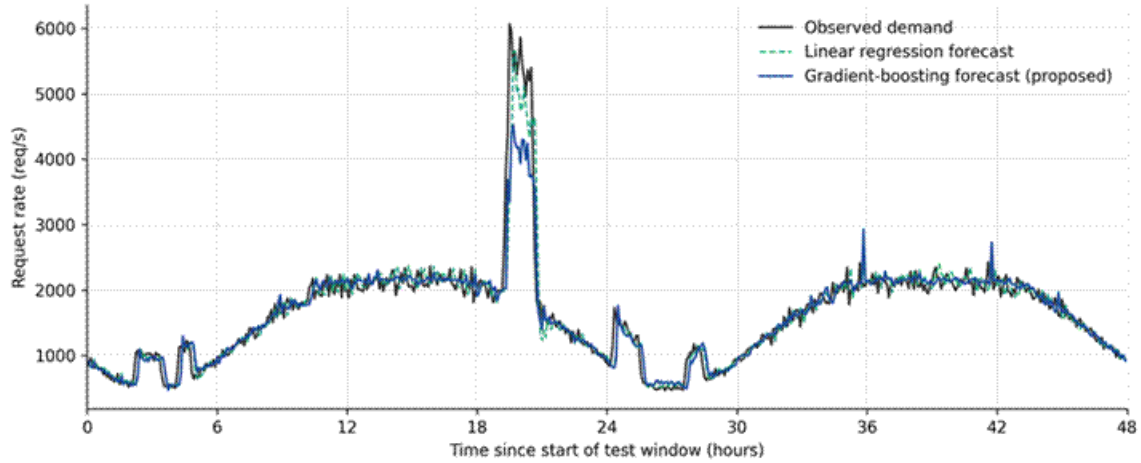


Fig. 2. Observed request rate and 10-minute-ahead forecasts over a 48-hour test window (first workload). The boosted model under-predicts the largest flash crowd.

6.2 Provisioning behaviour

Figure 3 follows a 24-hour window centred on the largest flash crowd of the first workload. Neither adaptive policy can avoid the provisioning delay. The reactive policy adds only a quarter of the pool per decision and needs several intervals to catch up, whereas the proposed controller reaches the required capacity in fewer decisions because the jump in observed demand activates the guard and the online correction. After the surge it releases surplus VMs within roughly two to three hours, while the cooldown-limited reactive policy carries idle capacity for many hours. The figure also exposes a cost of the design: after the forecast miss the correction drives the pool to approximately 250 VMs against about 150 needed, showing that protection against under-provisioning is bought with brief over-provisioning.

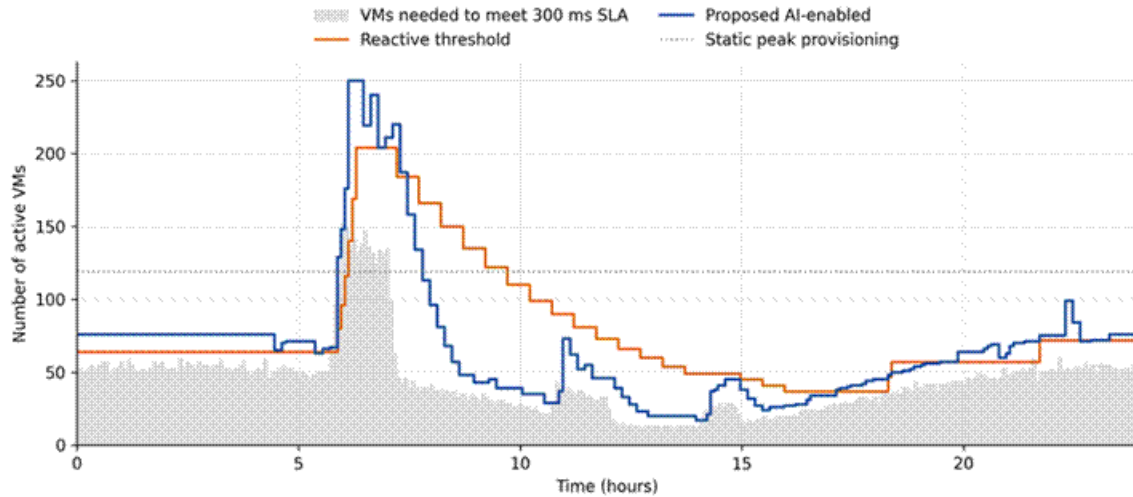


Fig. 3. Active VMs under the reactive, static and proposed policies against the VMs needed to satisfy the 300 ms SLA (first workload, 24-hour window around a flash crowd).

6.3 Cost and QoS comparison

Table 4 and Figure 4 report the aggregate results. Static provisioning sized for the training peak delivers the best QoS (0.11% violations) but is the most expensive option, at US\$ 258.7 per day and only 25.4% mean utilisation.

The reactive policy costs US\$ 139.1 per day but leaves utilisation at 47.7% and has a 95th-percentile latency of 202.8 ms. The proposed controller costs US\$ 117.0 per day, which is 15.9% below the reactive policy and 54.8% below static provisioning, while its SLA-violation rate (1.05%) is indistinguishable from that of the reactive policy (1.04%). Its 95th-percentile latency is 16.8% lower than the reactive policy's, and its share of dropped requests is marginally smaller (0.04% versus 0.05%).

Table 4. Performance of the four provisioning policies on the test days (mean $\pm$ SD over ten runs).

Metric	Static	Reactive	Predictive-LR	Proposed
Daily cost (US\$)	258.7 $\pm$ 16.9	139.1 $\pm$ 6.8	106.3 $\pm$ 1.2	117.0 $\pm$ 2.6
Mean utilisation (%)	25.4 $\pm$ 1.8	47.7 $\pm$ 1.4	61.4 $\pm$ 0.3	56.4 $\pm$ 0.4
SLA violations (% of intervals)	0.11 $\pm$ 0.34	1.04 $\pm$ 0.29	1.58 $\pm$ 0.45	1.05 $\pm$ 0.28
Mean latency (ms)	83.7 $\pm$ 3.1	132.9 $\pm$ 3.7	173.6 $\pm$ 5.6	148.7 $\pm$ 2.3
95th-percentile latency (ms)	104.9 $\pm$ 9.8	202.8 $\pm$ 5.8	205.5 $\pm$ 9.2	168.7 $\pm$ 2.3
Dropped requests (%)	0.00 $\pm$ 0.00	0.05 $\pm$ 0.04	0.08 $\pm$ 0.05	0.04 $\pm$ 0.02
Scaling actions per day	0.0 $\pm$ 0.0	20.2 $\pm$ 1.4	69.2 $\pm$ 4.0	66.3 $\pm$ 2.6

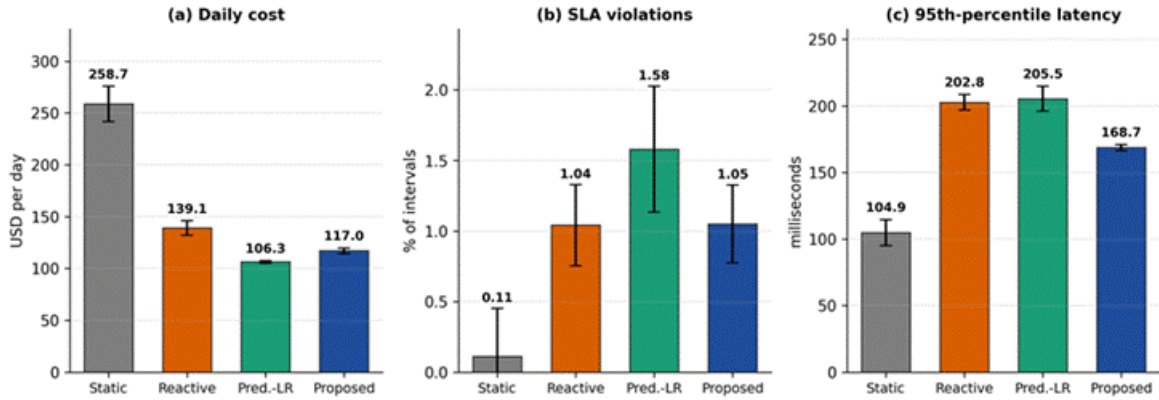


Fig. 4. Comparison of (a) daily cost, (b) SLA-violation rate and (c) 95th-percentile latency across policies (error bars: ± 1 SD over ten runs).

6.4 Discussion

Paired comparisons across the ten workloads support these observations. The proposed controller was cheaper than the reactive policy in all ten runs (paired t -test, $p < 0.001$) and had lower 95th-percentile latency in all ten, whereas the difference in SLA-violation rate was not statistically significant ($p = 0.94$). Against the linear-regression scaler, the proposed method spent about 10% more (US\$ 117.0 versus US\$ 106.3 per day) but cut SLA violations by roughly one third (1.05% versus 1.58%) and reduced them in every run. The linear-regression scaler thus saves money at the expense of QoS; the safeguards in Eq. (5) move the operating point towards a better balance.

Two practical observations follow. First, the savings come mainly from higher utilisation: the proposed controller keeps the pool 8.8 percentage points busier than the reactive one without paying in violations. Second, the gains are not free in operational terms. The proposed controller performs about 66.3 scaling actions per day compared

with 20.2 for the reactive policy, which matters on platforms with coarse billing or slow instance warm-up. Stronger scale-in hysteresis would reduce this churn at some cost in savings.

6.5 Limitations

Several limitations qualify these findings. The workloads are synthetic; although they contain the diurnal, weekly and bursty structure reported for production clusters [23], real traffic may differ in forecastability. The response-time model is a single-queue approximation that ignores multi-tier dependencies, heterogeneous instances and spot pricing. Tree-based forecasting cannot extrapolate beyond its training range, so unprecedented bursts rely on the correction and guard. Controller parameters were chosen on a validation window from the same generator as the test data, so sensitivity to distribution shift is unmeasured. Future work will therefore use production traces, quantile or recurrent forecasters, learned safety margins and an explicit energy model.

7. Conclusion

This paper examined how AI-based workload forecasting can be converted into provisioning decisions that improve Quality of Service and reduce computational cost. The proposed framework couples a gradient-boosted forecaster with a cost-aware controller that uses a validation-derived safety margin, online residual correction and scale-in hysteresis. In a simulation study on ten synthetic workloads, it cut daily cost by 15.9% relative to a reactive auto-scaler and by 54.8% relative to static peak provisioning, held the SLA-violation rate at the reactive level, lowered tail latency by 16.8% and improved on a linear-regression predictive scaler in QoS. The results also show that forecast accuracy alone is a poor guide to operational value: the controller that consumes the forecast determines whether accuracy becomes savings without harming users.

REFERENCES

1. Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing* (NIST Special Publication 800-145). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-145>
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, 53(4), 50–58. <https://doi.org/10.1145/1721654.1721672>
3. Masanet, E., Shehabi, A., Lei, N., Smith, S., & Koomey, J. (2020). Recalibrating global data center energy-use estimates. *Science*, 367(6481), 984–986. <https://doi.org/10.1126/science.aba3758>
4. Llorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). 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>
5. Qu, C., Calheiros, R. N., & Buyya, R. (2018). Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys*, 51(4), Article 73. <https://doi.org/10.1145/3148149>
6. Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015). Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15)* (Article 18). ACM. <https://doi.org/10.1145/2741948.2741964>
7. Rządca, K., Findeisen, P., Swiderski, J., Zych, P., Broniek, P., Kusmirek, J., Nowak, P., Strack, B., Witusowski, P., Hand, S., & Wilkes, J. (2020). Autopilot: Workload autoscaling at Google. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)* (Article 16). ACM. <https://doi.org/10.1145/3342195.3387524>
8. Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., ... Shen, H. (2018). A manifesto for future generation cloud computing: Research directions for the next decade. *ACM Computing Surveys*, 51(5), Article 105. <https://doi.org/10.1145/3241737>
9. Gill, S. S., Tuli, S., Xu, M., Singh, I., Singh, K. V., Lindsay, D., ... Garraghan, P. (2019). Transformative effects of IoT, blockchain and artificial intelligence on cloud computing: Evolution, vision, trends and open challenges. *Internet of Things*, 8, Article 100118. <https://doi.org/10.1016/j.iot.2019.100118>
10. Singh, S., & Chana, I. (2016). A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of Grid Computing*, 14(2), 217–264. <https://doi.org/10.1007/s10723-015-9359-2>
11. Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5), 755–768. <https://doi.org/10.1016/j.future.2011.04.017>
12. Calheiros, R. N., Masoumi, E., Ranjan, R., & Buyya, R. (2015). 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>

13. Kumar, J., Singh, A. K., & Buyya, R. (2021). Self directed learning based workload forecasting model for cloud resource management. *Information Sciences*, 543, 345–366. <https://doi.org/10.1016/j.ins.2020.07.012>
14. Xu, M., Song, C., Wu, H., Gill, S. S., Ye, K., & Xu, C. (2022). esDNN: Deep neural network based multivariate workload prediction in cloud computing environments. *ACM Transactions on Internet Technology*, 22(3), Article 75. <https://doi.org/10.1145/3524114>
15. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
16. Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5), 1189–1232. <https://doi.org/10.1214/aos/1013203451>
17. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.
18. Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)* (pp. 50–56). ACM. <https://doi.org/10.1145/3005745.3005750>
19. Tuli, S., Ilager, S., Ramamohanarao, K., & Buyya, R. (2022). Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks. *IEEE Transactions on Mobile Computing*, 21(3), 940–954. <https://doi.org/10.1109/TMC.2020.3017079>
20. Duc, T. L., García Leiva, R., Casari, P., & Östberg, P.-O. (2019). Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey. *ACM Computing Surveys*, 52(5), Article 94. <https://doi.org/10.1145/3341145>
21. Jassas, M. S., & Mahmoud, Q. H. (2022). Analysis of job failure and prediction model for cloud computing using machine learning. *Sensors*, 22(5), Article 2035. <https://doi.org/10.3390/s22052035>
22. Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., & Buyya, R. (2011). 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. <https://doi.org/10.1002/spe.995>
23. Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H., & Kozuch, M. A. (2012). Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)* (Article 7). ACM. <https://doi.org/10.1145/2391229.2391236>