International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604016

SmartScale: Real-Time AI-Driven Resource Optimization for Cloud Microservices

Manoj Kumar Reddy Kalakoti

Texas A&M University, Kingsville, USA

ABSTRACT: E-commerce platforms experience highly variable demand driven by promotions, flash sales, and seasonal peaks. Static provisioning wastes capacity during normal periods, while reactive autoscaling responds too slowly to sudden surges, creating a persistent trade-off between cloud cost and application reliability.

This study presents SmartScale, a closed-loop optimization framework for containerized microservices running on Kubernetes in AWS. SmartScale combines live infrastructure telemetry and business signals with short-horizon and seasonal demand forecasting to drive three actions: proactive replica scaling, resource request right-sizing, and reclamation of idle capacity. All actions pass through safety guardrails that enforce service level objectives (SLOs), disruption budgets, and canary validation with automatic rollback. The framework was evaluated on an e-commerce microservice platform against static provisioning and threshold-based reactive autoscaling.

Compared with the reactive baseline, SmartScale reduced monthly compute cost by 28-38%, decreased scaling response time from several minutes to under 60 seconds in anticipation of forecasted surges, and reduced SLO violation minutes by approximately 60% during peak events. Average cluster CPU utilization increased from 22-30% to 48-60%. No customer-facing incidents were attributable to automated reclamation actions.

Conclusions: Integrating demand forecasting with guarded, closed-loop actuation allows cloud-hosted e-commerce systems to reduce cost without sacrificing reliability. The framework provides a replicable pattern for organizations seeking to move from reactive capacity management toward predictive, policy-bound optimization.

KEYWORDS: Cloud resource optimization; Microservices; Demand forecasting; Autoscaling; Kubernetes right-sizing; FinOps; E-commerce; Service level objectives

I. OPENING CONTEXT AND FOUNDATION

1.1 The Cost and Reliability Trade-Off in Cloud Microservices

Cloud platforms promise elastic capacity, yet many organizations pay for substantially more capacity than they use. Surveys of cloud practitioners consistently report that a significant fraction of cloud spend is wasted on idle or over-provisioned resources [1]. The problem is acute in microservice architectures, where hundreds of independently deployed services each carry their own resource requests, replica counts, and scaling rules. Engineers tend to set these values conservatively at deployment time, because the cost of an outage is immediate and visible while the cost of over-provisioning is diffuse and delayed.

For e-commerce, the stakes are asymmetric. A checkout service that is under-provisioned during a promotional event directly loses revenue, and latency increases measurably reduce conversion rates. The natural response is to provision for peak load at all times, which leaves most capacity idle during normal operation. Reactive autoscalers narrow this gap but introduce their own limitations, because they act only after load has already increased and must wait for new pods and nodes to become ready.

This paper addresses the problem of how to reduce cloud cost for microservice platforms while maintaining or improving reliability, by combining live telemetry with demand forecasting to resize workloads and safely reclaim idle capacity. We present SmartScale, a closed-loop framework that integrates forecasting, right-sizing, and guarded actuation, and we evaluate it on cloud cost, scaling response time, and application reliability.

1.2 Characteristics of E-Commerce Workloads

E-commerce traffic exhibits several properties that complicate capacity management.

Strong periodicity. Demand follows daily and weekly cycles, with distinct peaks in evenings and weekends.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604016

Event-driven bursts. Marketing campaigns, flash sales, and holiday events create sudden surges that may exceed baseline traffic by an order of magnitude within minutes.

Heterogeneous services. Search, catalog, recommendation, cart, checkout, and payment services have different resource profiles, latency sensitivity, and criticality. A uniform scaling policy is therefore inappropriate.

Dependency propagation. An overloaded downstream service such as payments can cause cascading latency in upstream services, so scaling decisions for one service affect others.

Revenue sensitivity. Reliability failures translate directly into lost transactions, which makes safety a first-order requirement for any automated optimization.

1.3 Limitations of Reactive and Static Approaches

Static provisioning sets capacity based on historical peaks and rarely revisits it. Threshold-based autoscaling, such as scaling on CPU utilization, reacts to the current state of the system and has three well-known weaknesses. First, it lags behind demand because metrics must be collected, aggregated, and evaluated before action is taken, and new capacity then requires time to start. Second, utilization thresholds are poor proxies for user experience in services bound by queues, locks, or downstream calls. Third, reactive scaling addresses replica count but not the sizing of each replica, so resource requests remain inflated even as replicas scale.

1.4 Contributions

The contributions of this work are:
A closed-loop architecture that fuses infrastructure telemetry with business signals for demand forecasting at multiple horizons.
A capacity decision method that converts forecast uncertainty into replica and resource targets with explicit headroom.
A safe reclamation workflow with SLO-based gates, canary validation, and automatic rollback.
An evaluation on an e-commerce microservice platform, measuring cost, scaling response time, and reliability against static and reactive baselines.

II. BACKGROUND AND RELATED WORK

2.1 Autoscaling in Cloud Environments

Autoscaling research spans reactive, predictive, and hybrid approaches. A comprehensive taxonomy of autoscaling techniques classifies them by their analysis method, including threshold rules, queueing theory, control theory, reinforcement learning, and time-series analysis [2]. Reactive rule-based scaling is the most widely deployed approach, as in the Kubernetes Horizontal Pod Autoscaler, because it is simple and requires no model. Predictive approaches use forecasts of future load to provision capacity in advance and are particularly valuable when provisioning delays are significant [3].

2.2 Workload Forecasting

Forecasting methods for cloud workloads range from classical statistical models such as ARIMA and exponential smoothing to additive decomposition models that handle multiple seasonalities and holiday effects [4], and to deep learning models such as recurrent networks and attention-based architectures [5]. Evaluation of these methods on workload traces indicates that no single model dominates, and that model selection depends on the periodicity, noise level, and burstiness of the workload. For e-commerce, calendar information such as campaign schedules is a strong predictor that purely statistical models cannot learn from traffic history alone. A practical forecasting system therefore benefits from combining time-series features with exogenous business signals and from producing prediction intervals rather than point estimates.

2.3 Resource Right-Sizing

Production cluster studies show that requested resources frequently exceed actual use by a wide margin. Analysis of large-scale cluster traces and systems such as Google's Borg and Autopilot demonstrates that automatic adjustment of resource limits based on observed usage can substantially reduce slack while keeping the rate of resource-related failures low [6], [7]. In Kubernetes, the Vertical Pod Autoscaler provides recommendations of this kind, though it has historically required pod restarts to apply changes and has limited awareness of forecasted demand.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604016

2.4 Cloud Cost Management

The FinOps discipline frames cloud cost optimization as a continuous practice that combines visibility, allocation, and optimization actions such as rightsizing, commitment-based discounts, and elimination of unused resources [8]. A recurring finding is that visibility alone does not reduce cost unless it is connected to automated, trusted actuation. Engineers hesitate to act on recommendations when they cannot be sure that an optimization will not cause an outage, which is the central trust problem that this work addresses.

2.5 Reliability Engineering and Service Level Objectives

Site reliability engineering practice defines reliability through service level indicators, objectives, and error budgets [9]. An error budget quantifies how much unreliability a service may incur in a period, and it provides a rational basis for deciding when it is safe to take optimization risks. A framework that reclaims capacity should therefore be coupled to SLO state, becoming more conservative when the error budget is depleted.

2.6 Limitations of Existing Approaches

Existing solutions typically address one dimension of the problem. Reactive autoscalers do not anticipate demand. Predictive scalers often focus on replica count and ignore right-sizing and node-level reclamation. Right-sizing recommenders produce static recommendations without runtime safety validation. Cost tools report waste but do not act on it. SmartScale integrates forecasting, right-sizing, reclamation, and safety validation into one closed-loop system.

III. TECHNICAL FRAMEWORK AND PLATFORM DESIGN

3.1 Platform Context

The evaluated platform is an e-commerce system composed of approximately 40 microservices running on Amazon EKS across three Availability Zones. Core services include catalog, search, recommendation, cart, checkout, payment, and order management, backed by managed databases, caches, and message queues. Services are stateless where possible, which allows replica counts and resource requests to be changed without data migration.

3.2 SmartScale Architecture

SmartScale consists of five components.

Telemetry layer. Prometheus collects infrastructure and application metrics including CPU, memory, request rate, latency percentiles, error rate, and queue depth. OpenTelemetry traces provide service-level dependency and latency data. Business signals such as orders per minute, cart events, and a campaign calendar are ingested alongside technical metrics.

Streaming ingestion and feature store. Metrics and events are streamed through Kafka and stored as time series with derived features, including rolling statistics, hour-of-day, day-of-week, holiday and campaign indicators.

Forecasting engine. Two models operate at different horizons, described in Section 3.3.

Optimization and policy engine. This component converts forecasts and utilization profiles into scaling and sizing decisions, subject to safety guardrails described in Section 3.4.

Actuation layer. Decisions are applied through Kubernetes-native mechanisms: KEDA and the Horizontal Pod Autoscaler for replica counts, the Vertical Pod Autoscaler for resource requests, and Karpenter or the Cluster Autoscaler for node provisioning and consolidation.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604016

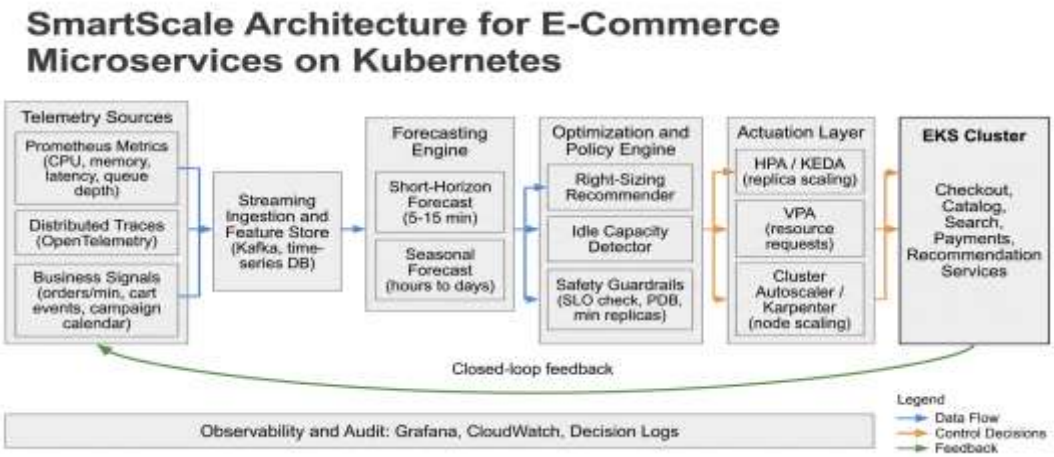


Figure 1. SmartScale architecture. Blue arrows indicate data flow, orange arrows indicate control decisions, and green arrows indicate closed-loop feedback.

3.3 Demand Forecasting

Two complementary forecasters were used.

Short-horizon forecaster (5 to 15 minutes). A gradient-boosted regression model predicts request rate per service using recent lag features, rate of change, and current business signals. This model is retrained frequently and optimized to detect the onset of surges quickly.

Seasonal forecaster (hours to days). An additive decomposition model captures daily and weekly seasonality and campaign effects. It supports pre-warming for scheduled events and informs reclamation decisions by identifying low-demand periods.

Both models output a point forecast y ^ t + h y ^ t+h and an upper prediction bound y ^ t + h u y ^ t+h u at confidence level 1 − α 1−α, so that capacity can be provisioned against the uncertainty of the forecast rather than only its mean. Forecast accuracy was assessed using the mean absolute percentage error and by the empirical coverage of the prediction interval.

3.4 Capacity Decision and Safety Guardrails

For each service, required replicas at horizon h h were computed from the upper forecast bound, the measured per-replica capacity μ μ at the SLO, and a target utilization ρ

ρ: N t + h = ⌈ y ^ t + h u ρ · μ ⌉ N t+h =⌈ ρ·μ y ^ t+h u ⌉

with N min ⁡ ≤ N t + h ≤ N max ⁡ N min ≤N t+h ≤N max enforced per service. The target utilization ρ ρ was reduced for latency-critical services such as checkout and payment, providing additional headroom. Resource requests for CPU and memory were set from observed usage percentiles plus a safety margin: R cpu = P 95 (u cpu) · (1 + m) R cpu =P 95 (u cpu)·(1+m) where m m is a margin set per service class, and memory requests were set higher, using P 99 P 99 with a larger margin, because memory exhaustion is not recoverable by throttling.

All proposed changes were evaluated against the following guardrails before actuation:

- SLO state check. Reclamation actions were suspended if the service's error budget burn rate exceeded a defined threshold.
- Disruption limits. Pod disruption budgets and minimum replica counts per service were always respected.
- Criticality tiers. Payment and checkout services used wider margins and required human approval for reductions beyond a set size.
- Event freeze. Reclamation was disabled during scheduled campaigns and a defined window before them.
- Rate limiting. Maximum change per interval and per day were enforced to prevent oscillation.
- Canary validation and rollback. Right-sizing changes were applied first to a subset of pods, evaluated against latency, error rate, and CPU throttling, and rolled back automatically upon violation.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604016

3.5 Validation and Testing Methodology

The framework was evaluated through a combination of production-derived replay and live operation. Three configurations were compared:

Static: Fixed replica counts and resource requests, set from historical peak load.
Reactive: CPU-threshold Horizontal Pod Autoscaler with default Cluster Autoscaler behavior.
SmartScale: Forecast-driven scaling, right-sizing, and guarded reclamation.

Evaluation included typical weekday and weekend traffic, and load-test-driven flash sale simulations with demand increasing by a factor of 8 to 10 within five minutes. Cost was computed from instance-hours, using on-demand pricing normalized across configurations. Scaling response time was defined as the interval from the onset of a demand surge, defined as the first minute where request rate exceeded 150% of the trailing baseline, to the moment when provisioned capacity reached the required level. Reliability was measured through SLO violation minutes, 99th percentile latency, error rate, and the number of incidents attributable to automated actions. Metrics were collected using Prometheus, CloudWatch, and Grafana.

IV. DEPLOYMENT MECHANICS

4.1 Predictive Scaling Control Loop

SmartScale operates as a control loop with five phases: observe telemetry, forecast demand with an uncertainty interval, decide capacity using the upper bound and target utilization, act through the autoscaling mechanisms, and verify the effect on SLOs and cost. Because new pods and nodes take time to become ready, the decision horizon is set at least as long as the sum of node provisioning, image pull, and application warm-up time. This ensures that capacity is ready before the demand arrives rather than after.

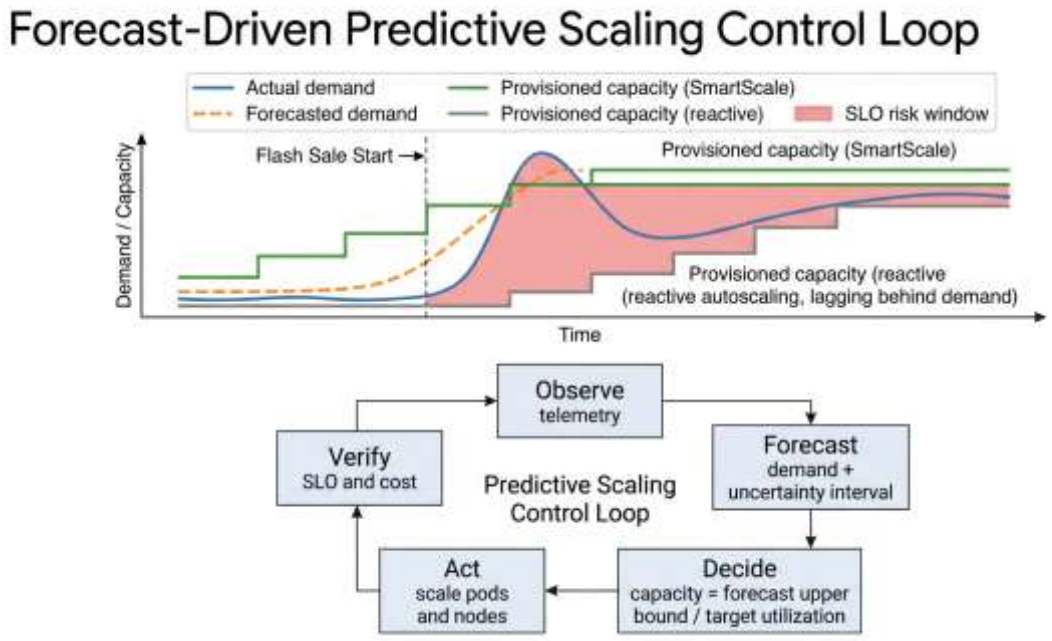


Figure 2. Forecast-driven scaling loop. Provisioned capacity is raised ahead of a forecasted surge, while reactive scaling lags behind demand and exposes an SLO risk window.

4.2 Safe Idle Capacity Reclamation

Reclamation proceeds in a gated workflow. A 14-day utilization profile is collected for each workload, and candidates are identified when utilization is persistently below a threshold relative to requests. Candidates then pass three gates. The first gate verifies that the forecasted peak demand, including its upper bound, remains below the reduced capacity with headroom. The second verifies that minimum replicas and disruption budgets are satisfied. The third applies the change to a canary subset of pods and evaluates latency, error rate, and throttling against the SLO. Only after the canary

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604016

passes does the change roll out to remaining pods in batches. Any gate failure blocks or reverses the change, logs the decision, and starts a cooldown period before the workload is re-evaluated.

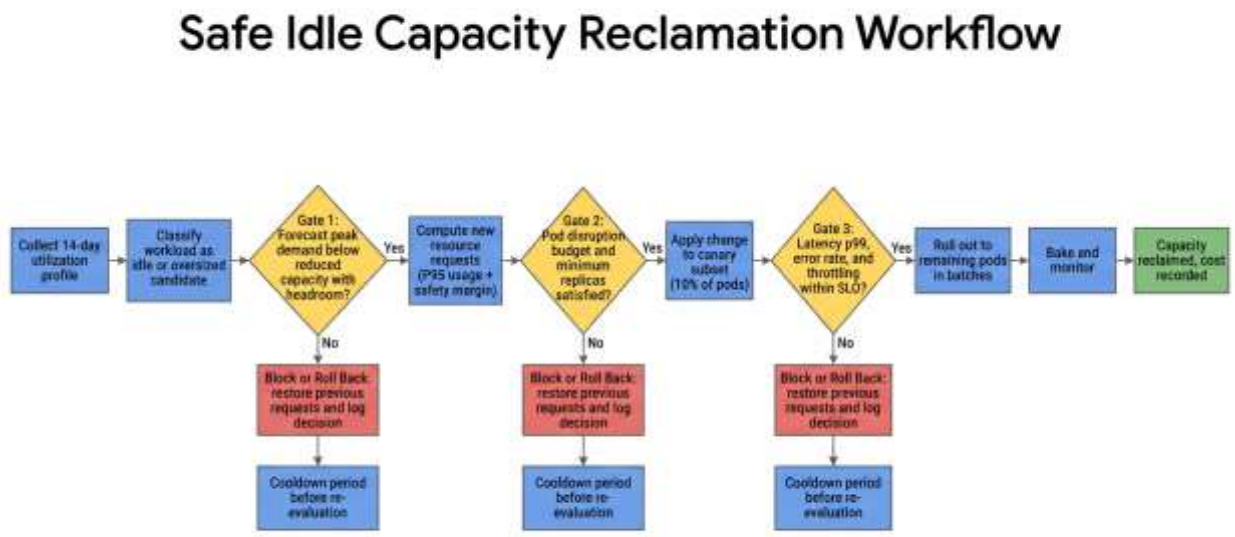


Figure 2. Safe reclamation workflow. Each gate has a rollback path that restores previous resource requests.

4.3 Node-Level Consolidation

After pods are resized, node capacity is consolidated. Underutilized nodes are cordoned and drained within disruption limits so that workloads are repacked onto fewer nodes, and Karpenter removes empty nodes. Instance type selection favors the lowest-cost types that satisfy the aggregate resource profile, and interruption-tolerant workloads such as batch and non-critical background services were placed on Spot capacity, while checkout and payment services remained on on-demand instances.

4.4 Observability and Auditability

Every SmartScale decision was recorded with its inputs, forecast values, guardrail evaluations, and outcome. Dashboards displayed forecast against actual demand, provisioned against required capacity, cost per order, and SLO burn rate. This record allowed engineers to audit and, where necessary, override decisions, which was a key factor in building trust in automated actions.

Table 1. Mapping of SmartScale Components to Cloud Mechanisms

SmartScale Function	Mechanism	Scope
Replica scaling	KEDA / HPA	Per service
Resource right-sizing	VPA with custom recommender	Per container
Node scaling and consolidation	Karpenter / Cluster Autoscaler	Cluster
Telemetry	Prometheus, OpenTelemetry	All services
Forecast storage and serving	Kafka, time-series store	Platform
Decision audit	Structured logs, Grafana	Platform

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604016

V. IMPLEMENTATION OUTCOMES AND MEASURED IMPACT

5.1 Forecast Accuracy

The short-horizon forecaster achieved a MAPE of 6-9% at the 10-minute horizon across core services under normal traffic, increasing to 12-18% during unscheduled surges. The seasonal forecaster achieved a MAPE of 8-12% at the 24-hour horizon, with the lowest error on services where campaign calendar features were available. The 90% prediction intervals achieved an empirical coverage of 86-92%, indicating that the uncertainty estimates were well calibrated for capacity decisions.

Table 2. Forecast Performance

Forecaster	Horizon	MAPE (normal)	MAPE (surge)	90% Interval Coverage
Short-horizon (gradient boosting)	10 min	6-9%	12-18%	86-90%
Seasonal (additive decomposition)	24 h	8-12%	15-22%	88-92%

5.2 Cost Reduction

Compared with static provisioning, SmartScale reduced compute cost by approximately 40-50%, and compared with the reactive baseline by 28-38%. Savings came from three sources: right-sizing of resource requests, which allowed denser node packing; scale-down of idle capacity during low-demand periods; and node consolidation with Spot usage for tolerant workloads. Average CPU utilization increased from 22-30% under static provisioning to 48-60% under SmartScale. Cost per thousand orders, a more meaningful efficiency metric for e-commerce, decreased in proportion.

5.3 Scaling Response Time

Under the flash sale simulations, reactive autoscaling required 4 to 9 minutes from surge onset to reach the required capacity, including metric lag, scaling decisions, node provisioning, and application warm-up. SmartScale provisioned capacity ahead of the surge when the event was forecasted or when the short-horizon model detected the early slope, so required capacity was available within 60 seconds of surge onset for most services. For unscheduled surges where the forecast could not anticipate the event, the response time was bounded by the short-horizon model's detection delay and was approximately 1 to 3 minutes.

Table 3. Performance Comparison Across Configurations

Metric	Static	Reactive	SmartScale
Relative monthly compute cost	100%	78-85%	52-60%
Avg CPU utilization	22-30%	35-42%	48-60%
Scaling response time (surge)	Not applicable	4-9 min	Under 60 s (forecasted)
SLO violation minutes (peak events)	Low (over-provisioned)	High	About 60% lower than reactive
p99 latency during surge	Stable	Degraded	Stable
Cost per 1,000 orders	100%	75-83%	48-58%

5.4 Reliability Outcomes

During flash sale simulations, the reactive configuration showed SLO violations in the minutes before new capacity became available, with elevated p99 latency and increased error rates in checkout and search. SmartScale reduced SLO violation minutes by approximately 60% compared with the reactive baseline, and its p99 latency during surges remained close to the static configuration. Static provisioning remained reliable but at a significantly higher cost. During the evaluation period, no customer-facing incidents were attributed to automated reclamation. The canary gate blocked a number of proposed reductions that would have caused CPU throttling, and these cases were concentrated in services with bursty, short-lived CPU spikes that percentile-based sizing alone underestimated.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604016

5.5 Effect of Guardrails

Ablation experiments disabled individual guardrails to measure their contribution. Disabling canary validation produced occasional resource-related regressions after right-sizing. Disabling the event freeze resulted in capacity reductions shortly before a promotional event, requiring emergency scale-up. Disabling rate limits caused oscillation between scale-up and scale-down in a small number of services. These results indicate that the guardrails are integral to the framework rather than optional safety additions, and that the cost savings achieved are attributable to automation that engineers could trust enough to leave enabled.

Table 4. Guardrail Ablation Results

Guardrail Disabled	Observed Effect	Severity
Canary validation	Post-resize CPU throttling and latency regressions	Moderate
Event freeze	Capacity shortfall before promotion	High
Rate limiting	Scaling oscillation (flapping)	Low to moderate
SLO state check	Reclamation during degraded periods	Moderate
Criticality tiers	Over-aggressive reduction of payment tier	High

5.6 Operational Observations

Engineering time spent on manual capacity reviews decreased, as right-sizing recommendations were applied automatically within guardrails. The decision audit log was valuable in building trust, because teams could review why each change occurred. Services with poor metric quality or with load driven by non-HTTP triggers required custom signals, such as queue depth, to avoid inaccurate sizing.

VI. ADVANCED EXTENSIONS AND FUTURE DIRECTIONS

6.1 Reinforcement Learning for Joint Cost and SLO Optimization

The current decision engine uses a deterministic policy driven by forecast upper bounds. Reinforcement learning approaches could learn policies that optimize a reward combining cost and SLO penalties directly, potentially discovering trade-offs that fixed margins miss. Safe reinforcement learning with constraint enforcement would be required to prevent exploration from violating production reliability targets.

6.2 Dependency-Aware Scaling

Scaling decisions for one service affect its upstream and downstream neighbors. Incorporating the service dependency graph from distributed traces would allow coordinated scaling, so that when checkout is predicted to surge, its dependencies such as payment and inventory are scaled in advance with appropriate ratios.

6.3 Carbon-Aware Optimization

Reducing idle capacity also reduces energy consumption. Extending the optimization objective to include carbon intensity of regions and time windows could allow flexible workloads to be shifted to periods and locations with lower emissions, aligning cost and sustainability objectives.

6.4 Limitations

This study evaluated an e-commerce platform with predominantly stateless services and strong periodic patterns. Workloads with stateful scaling constraints, highly irregular demand, or limited history may achieve smaller benefits and may require longer observation periods before reclamation is safe. Forecast quality directly affects both cost savings and safety, and sudden unprecedented events cannot be forecast. For this reason the reactive scaling path and the guardrails must remain active alongside the predictive path. Results should be validated across other platforms and traffic profiles.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 4, July- August 2024 |

DOI:10.15662/IJEETR.2024.0604016

VII. CONCLUSION

Cloud cost optimization and application reliability are often treated as opposing goals, but the tension largely originates in how capacity decisions are made. Static provisioning wastes resources, and reactive scaling responds too late. SmartScale addresses this by combining live telemetry with demand forecasting to provision capacity ahead of need, right-size workloads according to observed usage, and reclaim idle capacity through a gated, reversible workflow bound to service level objectives.

The measured outcomes of lower compute cost, faster scaling response, higher utilization, and fewer SLO violations show that predictive, policy-bound automation can improve efficiency and reliability together. Guardrails such as canary validation, event freezes, and criticality tiers were essential to making automation safe enough to be trusted in a revenue-critical e-commerce environment. The framework is replicable using standard Kubernetes and AWS components and provides a practical path from reactive capacity management toward predictive, auditable optimization. Future work includes safe reinforcement learning, dependency-aware coordinated scaling, and carbon-aware optimization.

VIII. AUTHOR STATEMENTS

Ethical approval: The conducted research is not related to either human or animal use.
Conflict of interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.
Acknowledgement: The authors declare that they have nobody or no company to acknowledge.
Author contributions: Manoj Kumar Reddy Kalakoti: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization, Project administration.
Funding information: The authors declare that there is no funding to be acknowledged.
Data availability statement: The configuration templates, forecasting pipelines, and benchmark results used in this study are available upon reasonable request. Production data has been anonymized to protect confidential business information.

REFERENCES

[1] AWS, (2023). Reliability Pillar - AWS Well-Architected Framework, Amazon Web Services. docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html.
[2] T. Lorido-Botran, J. Miguel-Alonso, J.A. Lozano, (2014). A review of auto-scaling techniques for elastic applications in cloud environments, J. Grid Comput. 12 (4) 559-592.
[3] N. Roy, A. Dubey, A. Gokhale, (2011). Efficient autoscaling in the cloud using predictive models for workload forecasting, in: 2011 IEEE 4th International Conference on Cloud Computing, IEEE, Washington, DC, 500-507.
[4] S.J. Taylor, B. Letham, (2018). Forecasting at scale, Am. Stat. 72 (1) 37-45.
[5] B. Lim, S.O. Arik, N. Loeff, T. Pfister, (2021). Temporal Fusion Transformers for interpretable multi-horizon time series forecasting, Int. J. Forecast. 37 (4) 1748-1764.
[6] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, (2015). Large-scale cluster management at Google with Borg, in: Proceedings of the Tenth European Conference on Computer Systems (EuroSys), ACM, Bordeaux, France, 1-17.
[7] K. Rzadca, P. Findeisen, J. Swiderski, et al., (2020). Autopilot: Workload autoscaling at Google, in: Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys), ACM, Heraklion, Greece, 1-16.
[8] J.R. Storment, M. Fuller, (2023). Cloud FinOps: Collaborative, Real-Time Cloud Financial Management, 2nd ed., O'Reilly Media, Sebastopol, CA.
[9] B. Beyer, C. Jones, J. Petoff, N.R. Murphy, (2016). Site Reliability Engineering: How Google Runs Production Systems, O'Reilly Media, Sebastopol, CA.