Proactive Auto-Scaling of Containerized Applications in Cloud Environment Using Ensemble-Based Workload Prediction

A Synopsis

submitted in partial fulfilment of the requirements of the degree of

Doctor of Philosophy

In

Computer / IT Engineering

Submitted By:

Trivedi Naimisha Shashikantbhai

[179999913019]

Supervisor:

Dr. Ajaykumar N. Upadhyaya

Professor

SAL Engineering & Technical Institute, SAL Education

Ahmedabad, Gujarat, India

Submitted to

GUJARAT TECHNOLOGICAL UNIVERSITY

AHMEDABAD, GUJARAT, INDIA

Contents

1 Abstract

Cloud-native applications increasingly rely on microservices and containerized deployment models to achieve scalability, flexibility, and efficient resource utilization. In such environments, application workloads are highly dynamic and may fluctuate rapidly due to bursty traffic, periodic demand cycles, or continuous growth in user requests. Managing resources elastically is therefore essential to maintain Quality of Service (QoS) and ensure reliable application performance. Traditional reactive autoscaling mechanisms, which provision additional container replicas only after predefined utilization thresholds are exceeded, are often limited by scaling lag. This delay can lead to transient Service Level Agreement (SLA) violations, increased latency, request rejection, and inefficient utilization of computing resources. This thesis presents a proactive autoscaling framework for containerized microservices that uses workload prediction to provision container replicas before demand surges occur.

The proposed framework integrates a clustering-enhanced hybrid ensemble workload prediction model with a MAPE-K — Monitor, Analyse, Plan, Execute, and Knowledge — control loop deployed on Docker Swarm. The hybrid prediction model combines K-means clustering, Prophet, Long Short-Term Memory (LSTM), and Linear Regression, using inverse-error-based weighting to capture seasonal, non-linear, and steady workload patterns. The prediction model is evaluated using RMSE, MAE, MAPE, and R^2 metrics, while autoscaling performance is assessed in terms of SLA compliance, latency, request rejection, throughput, replica count, and scaling actions. Experiments are conducted using two real workload traces which includes NASA HTTP logs and ClarkNet WWW traces and two synthetic workload patterns that have monotonically increasing and periodic traffic patterns to evaluate the effectiveness of the proposed framework.

The results show that the proposed proactive autoscaling framework consistently outperforms reactive autoscaling across all workload patterns. Compared with the reactive approach, it improves SLA compliance by approximately 5.3 percentage points and reduces mean P95 latency by 6.5–43.3%. Request rejection is also reduced by 68.2% for the NASA workload, 47.8% for ClarkNet, 9.9% for the increasing workload, and 31.2% for the periodic workload. These improvements are achieved with only a moderate resource overhead of approximately $\sim 3\%$ on average across all workload patterns compared with the reactive approach. The hybrid ensemble prediction model achieves an MAE of 2.492363, RMSE of 3.415395, and R^2 of 0.859265 on the NASA dataset, and the lowest MAE of 5.806198 and RMSE of 7.3380937 on the ClarkNet dataset among the compared baseline models. These results confirm that forecasting-based autoscaling reduces scaling delay, improves application performance, and supports efficient resource management in containerized microservice environments.

Keywords: Proactive autoscaling, container orchestration, Docker Swarm, workload prediction, hybrid ensemble model, LSTM, Prophet, K-means clustering, MAPE-K, SLA compliance.

2 Brief Description on the State of the Art of the Research Topic

2.1 Literature Review

Autoscaling in cloud and container environments has evolved from simple threshold-based reactive approaches to sophisticated machine learning (ML)-driven predictive systems. The following survey covers representative works spanning 2015–2026, organised by scaling strategy, virtualization technology, and monitoring granularity.

Reactive and Threshold-Based Approaches. Reactive autoscaling is the most common production-oriented approach. Calheiros et al. [1] proposed ARIMA-based workload prediction integrated with cloud autoscaling, demonstrating that even simple statistical forecasting improves QoS over purely reactive policies. Nguyen et al. [2] studied Kubernetes Horizontal Pod Autoscaler (HPA) for elastic container orchestration, highlighting its CPU-utilisation threshold mechanism and showing its limitations under bursty workloads. Huang et al. [3] specifically addressed Docker Swarm autoscaling for bursty workloads, proposing a hybrid reactive-predictive strategy. Reactive autoscaling is simple, interpretable, and easy to deploy, but it has several limitations. Thresholds are difficult to tune for all workloads. If thresholds are low, the system may over-scale; if thresholds are high, the system may violate SLA. Reactive methods also do not anticipate future workload. Therefore, they are vulnerable to sudden bursts and periodic peaks.

ML-Based Predictive Autoscaling. Proactive autoscaling uses CPU utilization, memory utilization or workload prediction to trigger scaling in advance. Iqbal et al. [4] proposed dynamic workload pattern prediction for proactive web application autoscaling, combining ARIMA with neural networks. Deep learning methods have become popular because they can capture temporal dependencies in workload sequences. Dang-Quang and Yoo [5] used Bidirectional LSTM for Kubernetes autoscaling, achieving significant latency reductions. Luo et al. [6] introduced workload learning-based microservice autoscaling, demonstrating that accurate short-horizon forecasts reduce oscillation and improve SLA compliance. Meng et al. [7] proposed multi-level ML-based burst-aware autoscaling for SLO assurance, addressing both gradual trends and sudden spikes.

Buchaca et al. [8] presented proactive container autoscaling for cloud-native ML services, combining time-series forecasting with container warm-up strategies. Abdullah et al. [9] extended proactive autoscaling to fog microdata centres using predictive control. Ahmad et al. [10] proposed predictive hybrid autoscaling for containerised applications, combining reactive and proactive policies. De

Souza et al. [11] used ML to predict and avoid SLA violations in containerised applications. Dogani et al. [12] proposed federated learning-based proactive autoscaling for edge containers.

Reinforcement Learning and Transformer-Based Approaches. Lipari et al. [13] proposed dynamic and forecast-based Kubernetes autoscaling using reinforcement learning. Wanigasooriya and Ekanayake [14] introduced NimbusGuard, a deep Q-network-based proactive Kubernetes autoscaler. Kumar et al. [15] proposed InformerAutoScale for proactive autoscaling using Informer transformers. Kumar et al. [16] presented a multivariate transformer-based MAPE autoscaling framework. Bâ et al. [17] leveraged transformer interpretability for proactive cloud resource scaling.

Ensemble and Hybrid Prediction Models. Ensemble methods outperform single models by improving robustness and generalization. However, standalone models such as LSTM fail to consistently handle diverse workload patterns, motivating the use of hybrid ensemble approaches.[18]. Shariffdeen et al. [19] demonstrated that ensemble forecasting (ARIMA + exponential smoothing + regression) yields significantly lower forecast errors than individual models. Nguyen et al. [20] showed that an ESN-based ensemble outperforms all component models across three diverse datasets. Saxena et al. [21] reported that ensemble learning improves prediction accuracy by 14.99–27.55% on RMSE and 22.57–76.86% on R^2. Singh et al. [22] explicitly stated: "*no prediction model is best in general*." Pandeeswari et al. [23] showed that a hybrid ARIMA-LSTM-ELM ensemble outperforms Prophet and standalone ARIMA-LSTM on all metrics. Guruge and Priyadarshana [24] demonstrated that a Prophet-LSTM hybrid achieves 69.4% lower MSE and 78.4% lower MAE than the best single-model Bi-LSTM on the NASA dataset. My published work, "Optimization Enabled Elastic Scaling in Cloud Based on Predicted Load for Resource Management," proposes a predictive elastic scaling approach for efficient cloud resource allocation. The method predicts workload demand using CPU, memory, and bandwidth parameters, and then performs optimized horizontal and vertical virtual machine scaling using the proposed Leader Harris Honey Badger Algorithm (LHHBA). LHHBA is a hybrid optimization algorithm that combines Leader Harris Hawks Optimization and the Honey Badger Algorithm to identify cost-effective scaling decisions while considering CPU, memory, and storage requirements. The work aims to reduce scaling cost, improve resource utilization, and support better cloud service performance under increasing workload demands.

2.2 Research Gap

Analysis of the surveyed literature reveals the following critical gaps that motivate the proposed research:

1. **Reactive Scaling Lag in Container Environments.** Most production systems, including Kubernetes HPA [2], rely on infrastructure-level monitoring metrics such as CPU-utilization

thresholds. Since container startup takes 30–90 seconds, reactive systems consistently under-provision during demand spikes, causing SLA violations.

2. **Lack of Accurate Prediction-Based Proactive Scaling for Multi-Pattern Workloads.** Existing predictive autoscalers use a single model (LSTM, ARIMA, or Prophet), which performs well only on specific workload patterns. Real-world workloads exhibit mixed seasonality, burstiness, and steady trends simultaneously. No prior work combines K-means workload clustering with an inverse-error-weighted ensemble of Prophet, LSTM, and Linear Regression for autoscaling.

3. **Lack of Integration with Real Deployment Systems.** Many studies validate prediction accuracy in isolation or use simulation. There is limited work that integrates prediction, monitoring, planning, and scaling execution in a real container orchestration testbed such as Docker Swarm.

4. **Insufficient Integration of Workload-Pattern-Aware Prediction with Real-Time System Constraints.** A practical autoscaler must translate predicted workload into replica count using per-replica capacity while considering current CPU, p95 latency, SLA threshold, replica bounds, and cooldown. This integration is still insufficiently addressed.

3 Problem Statement

3.1 Problem Description

Containerized cloud applications experience time-varying workloads that may change gradually, periodically, or suddenly. An autoscaling mechanism must determine the required number of replicas at the right time so that the application can satisfy QoS and SLA requirements without unnecessary resource usage. Static provisioning cannot adapt to workload variation. Reactive autoscaling is simple but it reacts only after current metrics cross thresholds. Therefore, reactive autoscaling suffers from scaling lag, temporary under-provisioning, high p95 latency, request rejection, and SLA violations during sudden workload growth. **Proactive autoscaling** addresses this by predicting future workload and provisioning replicas in advance. However, accurate multi-step workload forecasting for heterogeneous, non-stationary cloud traffic remains an open challenge. Existing single-model predictors fail to generalise across different workload patterns (seasonal, bursty, steady), and most proposals lack end-to-end validation in real containerised environments.

3.2 Formal Problem Formulation

Let $A(t) \in \mathbb{R}^+$ denote the observed request arrival rate (RPS) at time t, and let $\hat{A}(t+H)$ denote the predicted workload H minutes into the future. The workload predictor is defined as a forecasting function that uses the recent historical workload window $\hat{A}(t+H) = f_\theta(A(t-W+1), A(t-W+2), \ldots, A(t))$ where W is the historical observation window and f_θ is the trained forecasting model. Given the predicted workload, the number of required replicas is: $R^*(t) = \left\lceil \frac{\hat{A}(t+H)}{T_{\max}.\alpha} \right\rceil$ where $T_{\max}$ is the per-replica throughput capacity (RPS) and $\alpha \in (0,1)$ is a safety margin factor. The term $T_{\max}.\alpha$ represents the effective usable capacity of one replica. This prevents the autoscaler from assuming that each replica can continuously operate at its absolute maximum throughput.

The autoscaling objective is to determine a replica assignment policy $\pi : \mathcal{S} \to \mathcal{A}$ that minimises the expected SLA violation rate and reduce response latency while keeping the replica count within the available cluster capacity:

$$\min \left[\mathbb{E}\left(\frac{1}{T} \sum_{t=1}^{T} L_{95}(t) \right) + \eta \mathbb{E}\left(\frac{1}{T} \sum_{t=1}^{T} \mathbf{1}[L_{95}(t) > L_{\text{SLA}}] \right) \right]$$

$$\text{subject to} \quad R_{\min} \leq R^*(t) \leq R_{\max}, \quad \forall t$$

where $L_{95}(t)$ is the 95th-percentile response latency at epoch t, L_{SLA} is the SLA threshold, $R(t)$ is the average replica count, and $R_{\max}$ is the cluster capacity, S is the system state and A is the scaling action space.

The core research challenge is to construct an accurate workload predictor $\hat{A}(t+H) = f_\theta(A(t-W+1), \ldots, A(t))$ that minimises prediction error across heterogeneous workload patterns, and enables the autoscaler to provision replicas ahead of demand, thereby reducing scaling delay, lowers p95 response latency, minimizes SLA violations, and improves resource efficiency.

4 Objectives and Scope

4.1 Research Objectives

1. **Design a clustering-enhanced hybrid ensemble model** for accurate workload prediction in cloud environments, combining K-means clustering with Prophet, LSTM, and Linear Regression using inverse-error-based dynamic weighting.

2. **Develop a MAPE-K-based proactive autoscaling framework** that uses multi-step ahead forecast to provision container replicas before demand materialises, eliminating reactive scaling lag.

3. **Implement and deploy a real containerized testbed** with cluster on Docker Swarm architecture with Prometheus-based monitoring, cAdvisor container metrics, and trace-driven workload generation using Locust, enabling reproducible end-to-end evaluation.

4. **Evaluate and compare four autoscaling strategies** (no scaling, static provisioning, reactive, and proactive) across four workload patterns (NASA HTTP, ClarkNet HTTP, increasing, periodic) using SLA compliance, P95 latency, request rejection rate, and average replica count.

4.2 Scope of Work

Inclusions:

- Containerized microservices deployed on Docker Swarm (horizontal scaling only)
- CPU-bound stateless microservice as the target application
- Real-world HTTP workload traces (NASA, ClarkNet) and 2 synthetic workload patterns
- Application-level monitoring (RPS, P95 latency) and container-level monitoring (CPU, replicas)
- Offline model training with online MAPE-K control loop execution
- Comparison with reactive autoscaling as the primary baseline

Exclusions:

- Vertical scaling (CPU/memory resource limits per container) and node scaling
- Multi-tier microservice architectures with inter-service dependencies
- Online model retraining or concept drift adaptation during execution
- Memory-bound or I/O-bound workload types
- Kubernetes or other orchestration platforms
- Cost-based optimization with cloud pricing models

5 Original Contribution

The primary original contributions of this research are:

1. **Clustering-Enhanced Hybrid Ensemble Prediction Model.** A novel workload prediction architecture that applies K-means clustering on extracted features (variance, periodicity, rate-of-change) to classify workload patterns, then dynamically weights Prophet, LSTM, and Linear Regression predictions using per-window inverse-RMSE weighting. The model dynamically combines multiple predictors using cluster-aware and error-adaptive weighting, improving robustness for heterogeneous workload patterns for containerised autoscaling.

2. **Proactive Autoscaling Framework for Docker Swarm.** A complete closed-loop autoscaling system implementing the MAPE-K model with workload prediction, latency-awareness, stability controls (cooldown, confirmation counters). The framework is designed for Docker Swarm and validated on real cloud infrastructure, filling a gap in existing literature focused on Kubernetes.

3. **Open Experimental Testbed and Reproducible Evaluation Protocol.** A three-node AWS EC2 testbed with Docker Swarm, Prometheus, cAdvisor, and trace-driven Locust workload generation, providing a reproducible evaluation framework for future autoscaling research.

4. **Multi-Workload Benchmarking with Four Experimental Modes.** A systematic comparative evaluation of four provisioning strategies (Exp A: no scaling; Exp B: static; Exp C: reactive; Exp D: proactive) across four workload patterns (NASA, ClarkNet, increasing, periodic) on identical real infrastructure.

6 Methodology

6.1 Overview of the Proposed Framework

The architecture and workflow of the proposed proactive autoscaling framework deployed on a Docker Swarm cluster is illustrated in Figure 1. The diagram shows the complete flow from workload submission and request routing to service discovery, request handling, metrics collection, workload prediction, scaling decision, scaling trigger, replica scheduling, and updated container deployment. Client and external user requests are routed through the ingress/load balancer and handled by container replicas running on worker nodes. Runtime metrics collected through Prometheus and cAdvisor are used by the autoscaling policy to predict future workload demand, after which the autoscaler triggers scaling actions and the Swarm scheduler assigns updated replicas across worker nodes. The proposed methodology consists of two major models: a **workload forecasting model** and a **proactive autoscaling model**. The workload forecasting model predicts the future request rate using a

clustering-based hybrid ensemble of Prophet, LSTM, and Linear Regression. The proactive autoscaling model converts the predicted workload into a target number of container replicas before workload demand fully appears.

The framework is executed in two stages. In the first stage, historical workload traces from the NASA and ClarkNet web server logs, which are standard benchmark datasets, are used for model training. The workload traces are processed to extract request-rate time series, workload features are computed, workload patterns are identified using clustering, and the base forecasting models are trained. The trained models, cluster centroids, feature scalers, and model metadata are saved as artifacts.

In the second stage, the online autoscaling loop is executed inside the containerized environment. Prometheus and cAdvisor collect runtime metrics such as observed RPS, CPU utilization, p95 latency, and current replica count. The autoscaler uses recent RPS history and the saved forecasting artifacts to infer future workload. The proactive autoscaling model then computes the target replica count and Docker Swarm applies the scaling decision.

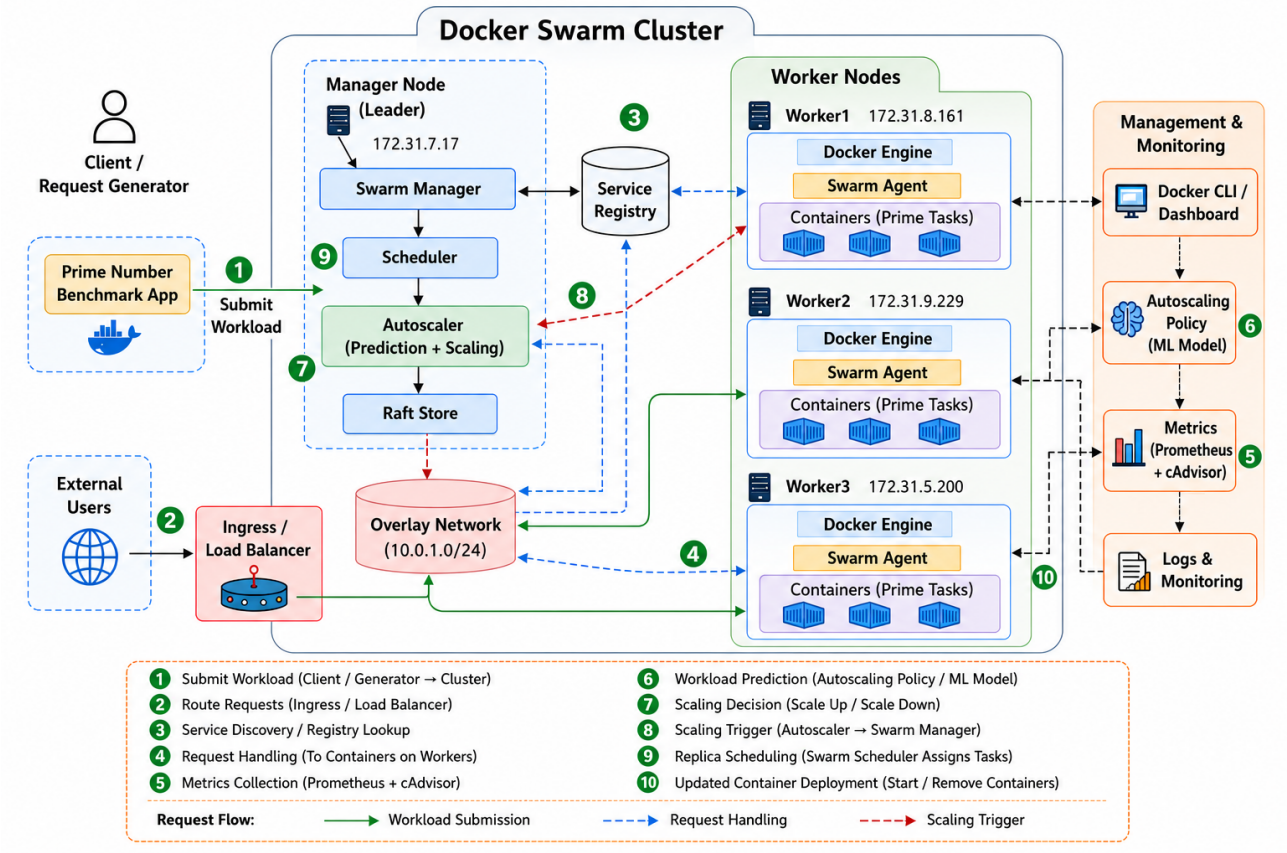


Figure 1: Architecture and workflow of the proposed framework in Docker swarm Cluster

The framework follows the MAPE-K control-loop concept internally. The Monitor phase collects runtime metrics, the Analyse phase performs workload inference using the saved hybrid ensemble forecasting model, the Plan phase computes the target replica count using the proactive autoscaling

model, and the Execute phase applies the scaling action through Docker Swarm. The knowledge component stores recent workload history, saved model artifacts, thresholds, cooldown state, and replica bounds.

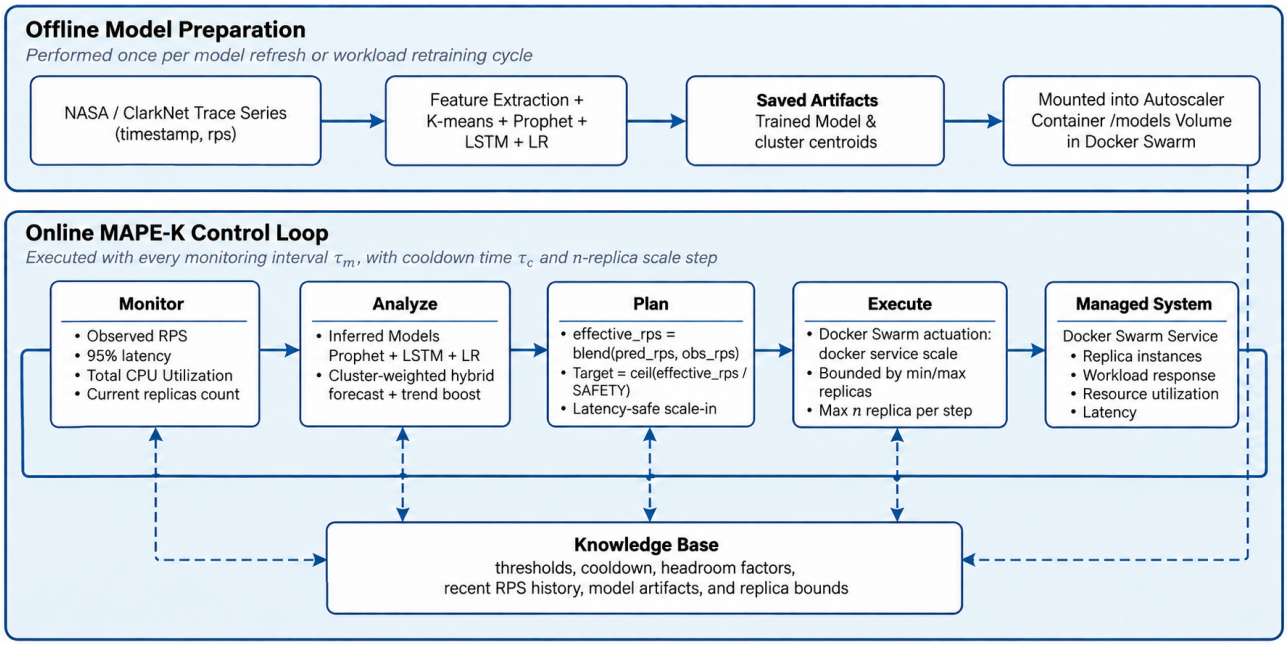


Figure 2: MAPE-K model for Proactive auto-scaling mechanism

6.2 Workload Forecasting Model

The workload forecasting model is designed to predict short-term future request demand in dynamic cloud environments. Cloud workloads may contain seasonal patterns, non-linear bursts, and steady trends [4]. Individual machine learning models are often inadequate for accurate workload prediction due to their inherent bias toward specific data distributions and limited ability to generalize across highly dynamic and multidimensional cloud environments [25]. A model that performs well in capturing long-term temporal dependencies may fail to effectively respond to abrupt, high-frequency fluctuations in workload patterns [26, 27]. Therefore, the proposed forecasting model combines workload feature extraction, clustering-based workload classification, and hybrid ensemble prediction.

Historical workload logs are first converted into timestamped request-rate series. Let $A(t)$ the observed request rate at time t. For each sliding workload window, three features are extracted: variance, periodicity, and rate of change. These features are used to represent the dominant behavior of the workload. The variance feature captures workload volatility and is computed as follows: $\text{Var}_t = \frac{1}{h-1}\sum_{i=t-h+1}^{t}(A(i)-\bar{A}_t)^2$, where h is the window size and $\bar{A}_t$ is the mean request rate within the window. The periodicity feature captures seasonal repetition using the autocorrelation function: $\text{Per}_t = \max_{1\le k\le p}|\text{ACF}(k)|$, where p denotes the maximum lag considered for periodicity detection. The rate-of-change feature captures sudden increase or decrease in workload: $\text{Roc}_t = \frac{A(t)-A(t-1)}{A(t-1)+\epsilon}$,

where ϵ is a small constant used to avoid division by zero. These features form the workload feature vector: $\mathbf{f}_t = [\text{Var}_t, \text{Per}_t, \text{Roc}_t]^T$. K-means clustering is then applied to the feature vectors to group workload windows into different workload behavior types. In this work, $K = 3$ is used to represent three major workload patterns: seasonal, non-linear, and steady. The K-means objective is: $J = \sum_{i=1}^{K} \sum_{\mathbf{f}_t \in S_i} \|\mathbf{f}_t - \mathbf{c}_i\|^2$, where S_i is the set of workload windows assigned to cluster i, and $\mathbf{c}_i$ is the centroid of cluster i. Workload windows with high periodicity are associated with seasonal behavior, windows with high variance or rate of change are associated with non-linear or bursty behavior, and windows with low variation are associated with steady behavior.

For a given workload window, the distance from the feature vector to each cluster centroid is computed as: $d_i(t) = \|\mathbf{f}_t - \mathbf{c}_i\|^2$.

The corresponding cluster-proximity weight is computed using inverse-distance weighting: $w_i(t) = \frac{\left(d_i(t)+\epsilon\right)^{-1}}{\sum_{j=1}^{K}\left(d_j(t)+\epsilon\right)^{-1}}$. These weights indicate the degree to which the current workload window belongs to each workload pattern. In parallel, performance-based weights are computed using recent prediction errors. A rolling buffer of the last $M = 10$ windows is maintained to compute the Root Mean Squared Error (RMSE) for each model. The inverse-error weights are: $w_{t,k}^{\text{error}} = \frac{\left(\text{RMSE}_{t,k}+\epsilon\right)^{-1}}{\sum_j \left(\text{RMSE}_{t,j}+\epsilon\right)^{-1}}$.

To balance workload pattern relevance and recent predictive performance, the final ensemble weights are computed as $w_{t,k} = \alpha\, w_k^{\text{cluster}}(t) + (1-\alpha)\, w_{t,k}^{\text{error}}$, where $\alpha \in [0, 1]$ controls the contribution of each component and $\sum_k w_{t,k} = 1$. The prediction layer employs three complementary models to capture diverse workload patterns. Prophet models seasonal and trend components using an additive formulation $P_{\text{Prophet}}(t) = g(t) + s(t) + \epsilon_t$, where $g(t)$ represents the trend and $s(t)$ captures seasonality. LSTM models non-linear temporal dependencies using gated recurrent units $h_t = \text{LSTM}(x_t, h_{t-1})$, where h_t represents the hidden state capturing temporal patterns. Linear Regression (LR) captures steady patterns using a linear relationship over lagged inputs: $P_{\text{LR}}(t) = \beta_0 + \sum_{i=1}^{w} \beta_i x_{t-i}$.

This combination of models ensures effective handling of seasonal, non-linear, and stable workload behaviors while maintaining computational efficiency. Let $P_{\text{Prophet}}(t)$, $P_{\text{LSTM}}(t)$, and $P_{\text{LR}}(t)$ denote the predictions of the three models. The final workload forecast is $\widehat{A}(t+H) = \sum_{k \in \{\text{Prophet,LSTM,LR}\}} w_{t,k} \cdot P_k(t)$. This hybrid weighting mechanism dynamically adjusts model contributions based on both workload characteristics and recent prediction accuracy, improving robustness across seasonal, non-linear, and steady workload patterns.

The forecasting model is prepared offline. During offline preparation, the training data is divided into training, validation, and testing sets. The base models are trained and evaluated, and the required artifacts are saved. These artifacts include the trained Prophet model, trained LSTM model, trained Linear Regression model, cluster centroids, feature scaling parameters, and ensemble metadata. Dur-

ing online autoscaling, these artifacts are loaded by the autoscaler container and used for inference. The online phase uses recent RPS history along with the saved artifacts to infer $\widehat{A}(t+H)$ without retraining the forecasting models.

6.3 Proactive Autoscaling Model

The proactive autoscaling model receives the predicted workload from the forecasting model and converts it into a target number of replicas. The purpose of the model is to provision replicas ahead of demand so that scaling delay is reduced and p95 latency remains within the SLA threshold.

The raw predicted workload may fluctuate because of short-term noise. Therefore, the prediction is first smoothed using exponential smoothing $\widehat{A}_s(t) = \lambda\widehat{A}(t+H) + (1-\lambda)\widehat{A}_s(t-1)$, where $\widehat{A}_s(t)$ is the smoothed prediction and λ is the prediction smoothing factor.

The smoothed prediction is then combined with the current observed workload

$A_{\text{eff}}(t) = \beta\widehat{A}_s(t) + (1-\beta)A(t)$, where $A_{\text{eff}}(t)$ is the effective workload used for scaling, and β controls the contribution of predicted demand relative to current observed demand. The required number of replicas is then computed using the derated per-replica capacity $R^*(t) = \left\lceil \frac{A_{\text{eff}}(t)}{\alpha T_{\max}} \right\rceil$, where $T_{\max}$ is the maximum RPS capacity of one replica and α is the safety factor. The term $\alpha T_{\max}$ represents the effective usable capacity of one replica. This avoids assuming that a replica can continuously operate at its absolute maximum capacity. The replica count is bounded by the minimum and maximum replica limits $R^*(t) = \min\left(\max(R^*(t), R_{\min}), R_{\max}\right)$.

The proactive autoscaling model also applies stability controls. Cooldown prevents repeated scaling on stale metrics. Downscale confirmation prevents premature scale-in. The latency guard blocks scale-in when the p95 latency is already above the SLA threshold.

Algorithm 1: Proactive RPS-Driven Autoscaling Algorithm

Input: Current replicas $R(t)$, observed RPS $A(t)$, predicted RPS $\hat{A}(t+H)$, previous smoothed prediction $\hat{A}_s(t-1)$, p95 latency $L_{95}(t)$, per-replica capacity $T_{\max}$, safety factor α, prediction smoothing factor λ, blending factor β, minimum replicas $R_{\min}$, maximum replicas $R_{\max}$, cooldown interval τ_c, last scale time t_{last}, latency SLA threshold L_{SLA}, downscale confirmation counter $v_d(t-1)$, required confirmations k_d

Output: Target replicas $R^*(t)$

if $t - t_{last} < \tau_c$ **then**
  return $R(t)$; `// Skip scaling during cooldown`
$\hat{A}_s(t) \leftarrow \lambda \hat{A}(t+H) + (1-\lambda)\hat{A}_s(t-1)$; `// Smooth prediction`
$A_{\text{eff}}(t) \leftarrow \beta \hat{A}_s(t) + (1-\beta)A(t)$; `// Blend predicted and observed workload`
$R^*(t) \leftarrow \left\lceil \frac{A_{\text{eff}}(t)}{\alpha T_{\max}} \right\rceil$; `// Compute required replicas`
$R^*(t) \leftarrow \min(\max(R^*(t), R_{\min}), R_{\max})$; `// Apply replica bounds`
if $R^*(t) < R(t)$ ***and*** $v_d(t-1) < k_d$ **then**
  $R^*(t) \leftarrow R(t)$; `// Block premature scale-in`
if $R^*(t) < R(t)$ ***and*** $L_{95}(t) > L_{SLA}$ **then**
  $R^*(t) \leftarrow R(t)$; `// Block scale-in during SLA violation`
return $R^*(t)$

The reactive baseline computes the target replica count from the ratio between current CPU utilization and target CPU utilization. If $U(t) > U_{\text{target}}$, the replica count increases proportionally; if $U(t) < U_{\text{target}}$, the replica count decreases proportionally. The final value is bounded by $R_{\min}$ and $R_{\max}$ with $R^*_{\text{reactive}}(t) = \left\lceil R(t) \times \frac{U(t)}{U_{\text{target}}} \right\rceil$.

6.4 Implementation Workflow

The complete implementation connects the workload forecasting model and the proactive autoscaling model inside an online control loop. The workload forecasting model is prepared offline and saved as a set of artifacts. These artifacts are mounted into the autoscaler container. During online execution, the autoscaler periodically queries Prometheus to obtain the current RPS, CPU utilization, p95 latency, and current replica count.

The recent RPS history is passed to the inference script, which loads the saved hybrid ensemble artifacts and predicts the future workload. The predicted workload is then passed to the proactive autoscaling model. The controller computes the target replica count, applies stability controls, and sends the final scaling command to Docker Swarm. After each scaling action, the cooldown state is

updated and the next control interval begins.

The overall implementation workflow can be summarized as Prometheus Metrics $\rightarrow$ Recent RPS History $\rightarrow$ Hybrid Ensemble Inference $\rightarrow$ Proactive Replica Decision $\rightarrow$ Docker Swarm Scaling.

This methodology therefore combines a clustering-based workload forecasting model with a proactive autoscaling model to provision replicas ahead of demand, reduce scaling delay, lower p95 latency, and minimize SLA violations.

7 Experimental Setup

7.1 Cluster Infrastructure

The experimental platform is deployed on Amazon EC2 instances configured as a three-node Docker swarm cluster. A Swarm cluster with a leader node acting as the Swarm Manager and two worker nodes, where all nodes are equipped with Intel Xeon E5-2686 v4 CPUs, 2 vCPUs, and 4 GB RAM; the leader node runs Docker Engine version 25.0.6, while worker1 and worker2 run Docker Engine version 20.10.7.

The target application is a CPU-intensive Flask/Gunicorn microservice that computes primes up to $n = 50{,}000$ per request, using trial-division-based approach with $O(n\sqrt{n})$ complexity and ≈400 ms CPU time per request on t3.small. Per-replica capacity $\mu \approx 10.4$ RPS. Capacity Profiling Algorithm used to measure per-replica capacity. Monitoring uses Prometheus v2.12.0 (15-second scrape) and cAdvisor v0.25.0. The autoscaling control loop executes every 10 seconds with a 60-second cooldown.

7.2 Hybrid Ensemble Model Configuration

NASA HTTP dataset spans 31 days with 44,640 samples and is characterized by bursty and irregular workload patterns, while the ClarkNet WWW dataset covers 7 days with 8,899 samples and exhibits high-frequency oscillatory behavior; both benchmark datasets are used to train the prediction model. The prediction model is configured with $K = 3$ clusters (periodic, bursty, trending), and base models: Prophet (additive decomposition), LSTM (2 layers, 64 units, dropout 0.2), and Linear Regression (ridge, $\lambda = 0.01$). Ensemble weights use inverse-RMSE on per-cluster validation performance.

7.3 Autoscaling Controller Parameters

Proactive autoscaling controller parameters can be found in Table 1.

Table 1: Autoscaling Controller Parameters

Parameter	**Value**	**Description**
L_{SLA}	200 ms	P95 latency SLA threshold
H	10 min	Prediction lookahead horizon
W	60 min	Input window for workload prediction
τ_c	60 s	Minimum time between scaling actions
α	0.85	Safety factor for derating per-replica capacity
λ	0.7	Prediction smoothing factor
β	0.7	Predicted-versus-observed workload blending factor
k_d	3 cycles	Confirmation threshold for scale-in
$R_{\min}$	1	Minimum replica count
$R_{\max}$	12	Maximum replica count
Control interval	10 s	Autoscaling loop frequency

7.4 Experimental Conditions

Four experimental conditions are evaluated under identical infrastructure:

- **Exp A**: No scaling (single replica throughout)
- **Exp B**: Static provisioning (fixed 4 replicas)
- **Exp C**: Reactive autoscaling (CPU threshold: scale-up at 70%, scale-down at 30%)
- **Exp D**: Proactive autoscaling (hybrid ensemble prediction + MAPE-K controller)

8 Results and Comparisons

8.1 Evaluation Metrics

Performance of the hybrid ensemble workload prediction model is evaluated using four key metrics: MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), MAPE (Mean Absolute Percentage Error), and R^2 (coefficient of determination). Autoscaling performance is evaluated using SLA compliance (%), mean P95 latency (ms), completed and rejected requests, average replica count, and total scaling actions.

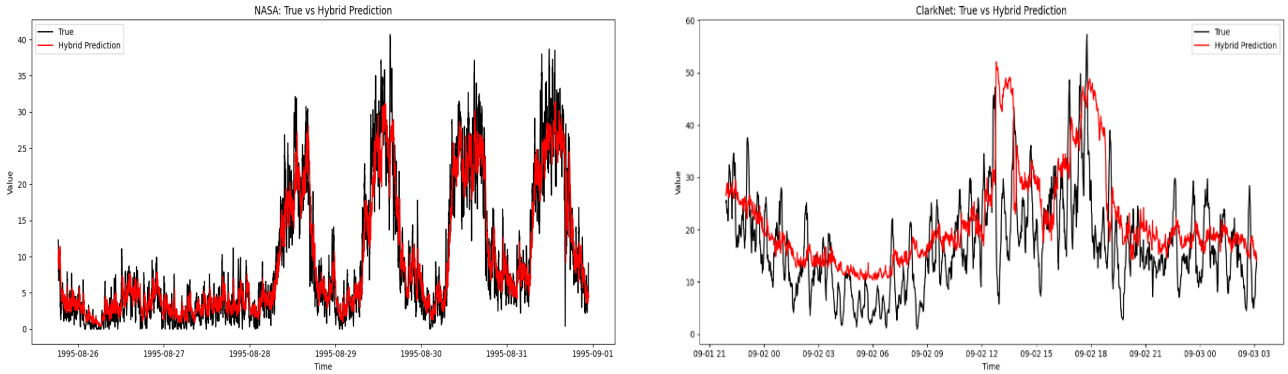
8.2 Prediction Model Performance

Table 2: Prediction Model Performance on NASA HTTP Dataset (W=60 min, H=10 min)

Model	**MAE**	**RMSE**	**MAPE**	R^2
Only LR	2.512523	3.466961	2.56E+06	0.855353
SVR	2.517843	3.539463	2.23E+06	0.849240
Only LSTM	2.556482	3.545775	3.65E+06	0.848702
TCN	2.725441	3.704726	3.85E+06	0.834833
Only Prophet	4.499269	5.891381	4.74E+06	0.582318
ARIMA	8.119328	10.05128	1.55E+07	−0.21578
Hybrid (Proposed)	**2.492363**	**3.415395**	**2.93E+06**	**0.859265**

Table 3: Prediction Model Performance — ClarkNet WWW Dataset (W=60 min, H=10 min)

Model	**MAE**	**RMSE**	**MAPE**	R^2
Only LR	5.811724	7.355730	52.654869	0.317038
Only LSTM	5.995649	7.530014	57.850128	0.284291
TCN	5.928095	7.547170	57.118645	0.281026
SVR	6.202307	7.756597	62.893710	0.240570
ARIMA	9.965972	11.716713	123.192620	−0.732833
Only Prophet	17.74555	22.873599	146.507280	5.604109
Hybrid (Proposed)	**5.806198**	**7.3380937**	**52.901305**	**0.307553**



(a) NASA (b) Clarknet

Figure 3: Actual vs. Predicted RPS

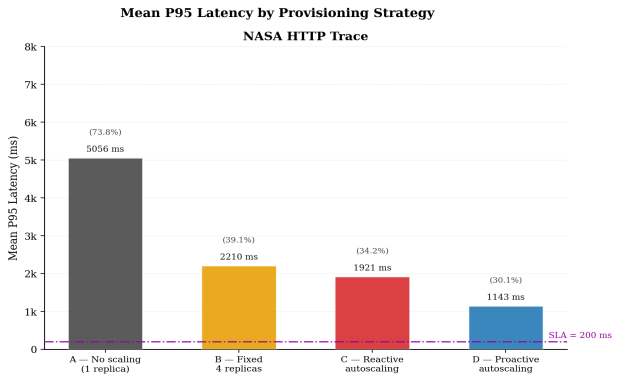
8.3 Autoscaling Performance Results

Table 4: Performance Summary — All Provisioning Strategies and Workload Patterns

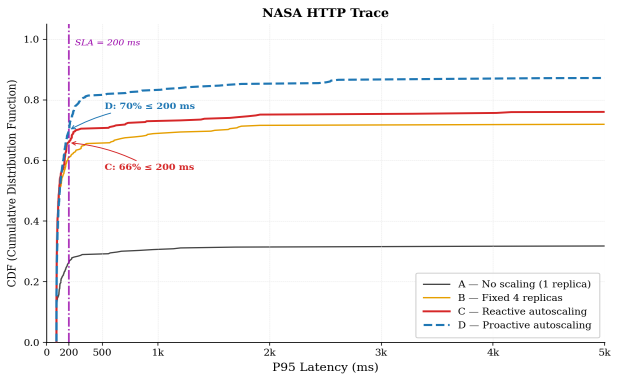
Workload	Exp	Completed	Rejected	SLO Miss	SLO Miss%	Max Tput	Avg Rep	Scale Act
NASA HTTP	A	25,182	58,813	270	73.8%	8.4	1.00	0
	B	68,402	15,594	143	39.1%	33.6	4.00	0
	C	66,162	17,834	125	34.2%	57.2	6.80	25
	D	**78,332**	**5,663**	**93**	**25.4%**	**58.8**	**7.00**	**21**
ClarkNet HTTP	A	19,640	47,137	226	89.7%	8.4	1.00	0
	B	56,664	10,112	120	47.6%	33.6	4.00	0
	C	57,687	9,090	71	28.2%	58.8	7.80	24
	D	**62,030**	**4,746**	**57**	**22.6%**	**62.2**	**7.60**	**21**
Increasing	A	29,604	137,856	346	96.1%	8.4	1.00	0
	B	101,326	66,135	263	73.1%	33.6	4.00	0
	C	167,461	3312	70	19.4%	93.2	8.03	16
	D	**167,461**	**2985**	**56**	**15.6%**	**93.2**	**8.19**	**13**
Periodic	A	27,353	126,073	313	86.9%	8.4	1.00	0
	B	91,254	62,172	236	65.6%	33.6	4.00	0
	C	143,836	9,590	143	39.7%	86.2	7.10	18
	D	**146,827**	**6,599**	**132**	**36.7%**	**86.2**	**7.80**	**15**

8.4 Latency Distribution and Visual Results

Figures 4–7 present bar charts of mean P95 latency and CDF plots of latency distribution for all four workloads.

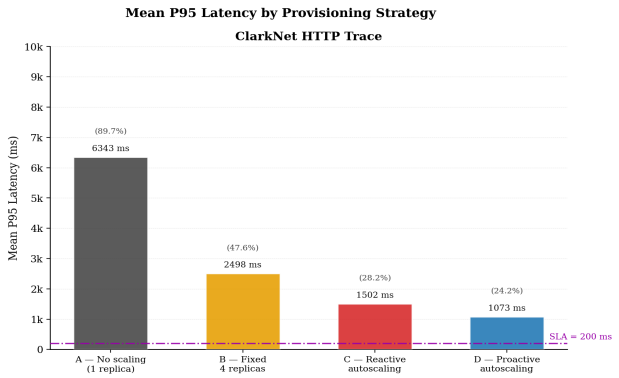


(a) NASA: Mean P95 Latency per Experiment

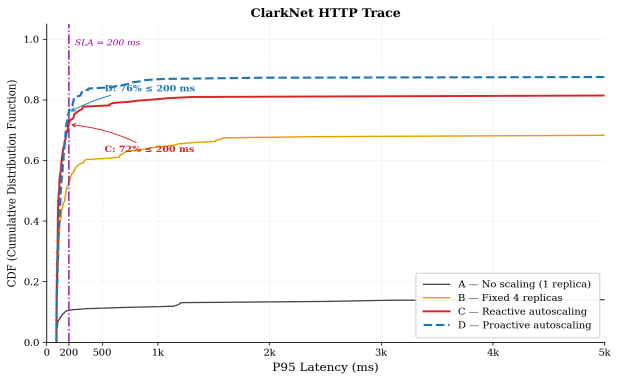


(b) NASA: CDF of P95 Latency

Figure 4: NASA HTTP Workload: Latency comparison across Exp A–D.

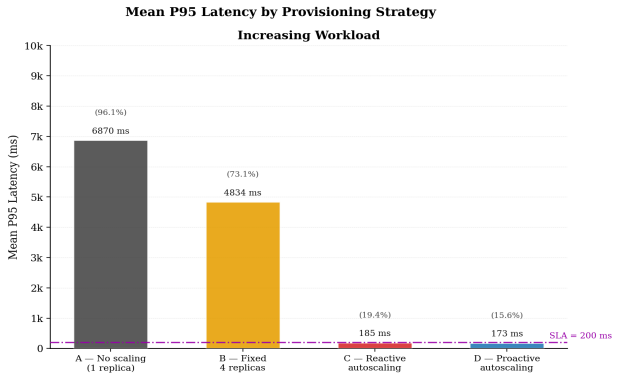


(a) ClarkNet: Mean P95 Latency per Experiment

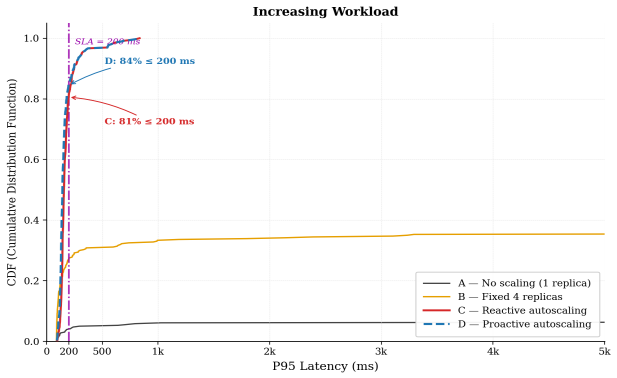


(b) ClarkNet: CDF of P95 Latency

Figure 5: ClarkNet WWW Workload: Latency comparison across Exp A–D.

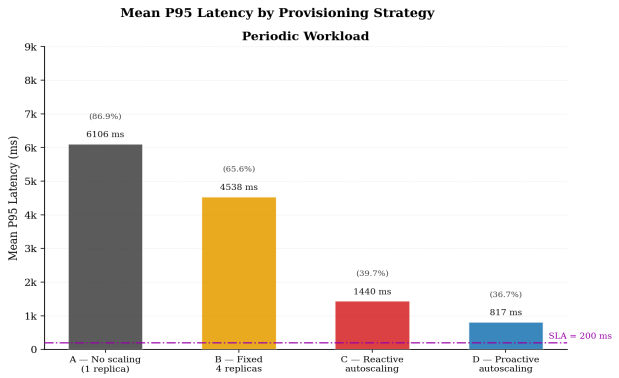


(a) Increasing: Mean P95 Latency per Experiment

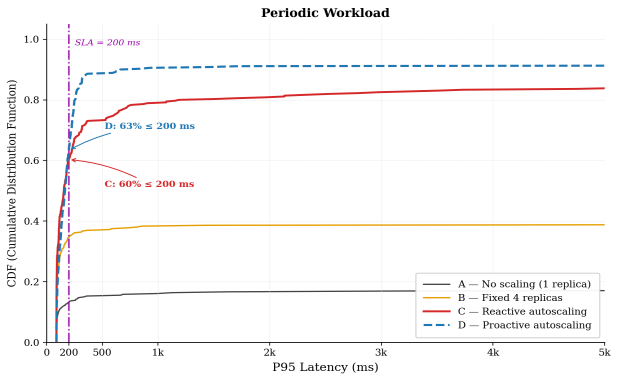


(b) Increasing: CDF of P95 Latency

Figure 6: Increasing Synthetic Workload: Latency comparison across Exp A–D.



(a) Periodic: Mean P95 Latency per Experiment



(b) Periodic: CDF of P95 Latency

Figure 7: Periodic Synthetic Workload: Latency comparison across Exp A–D.

8.5 Replica Count and RPS Dynamics

Figures 8–11 present Container replica count with arrival rate for Reactive (C) and Proactive (D) for all four workloads.

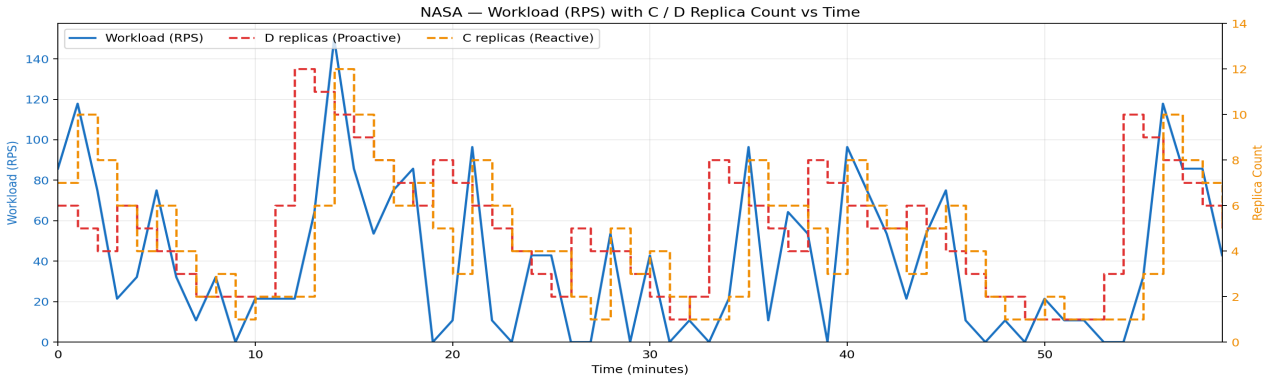


Figure 8: Container replica count with arrival rate for Reactive (C) and Proactive (D) — NASA HTTP Trace

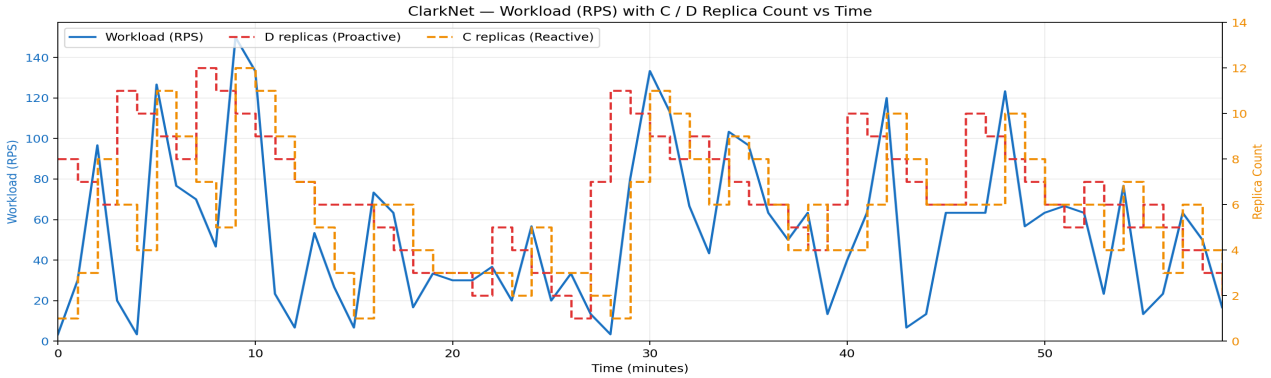


Figure 9: Container replica count with arrival rate for Reactive (C) and Proactive (D) — ClarkNet HTTP Trace

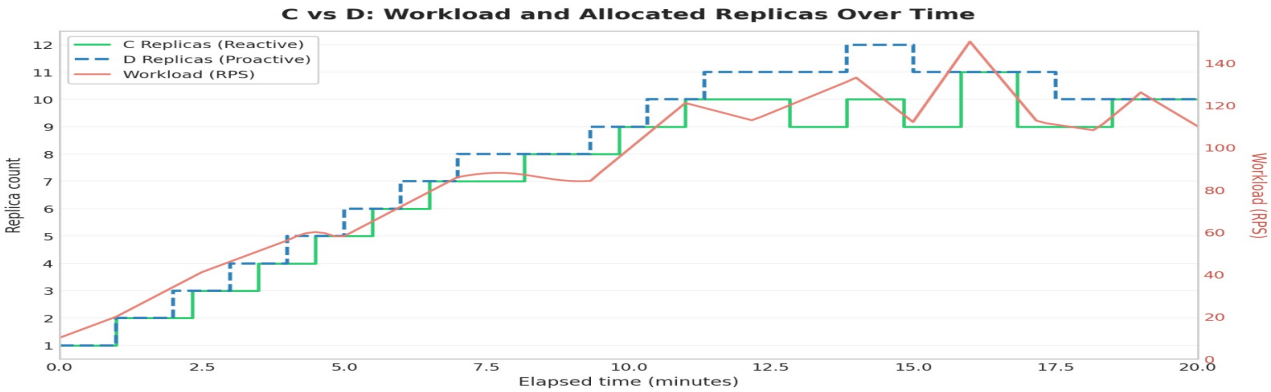


Figure 10: Container replica count with arrival rate for Reactive (C) and Proactive (D) — Monotonically Increasing Workload

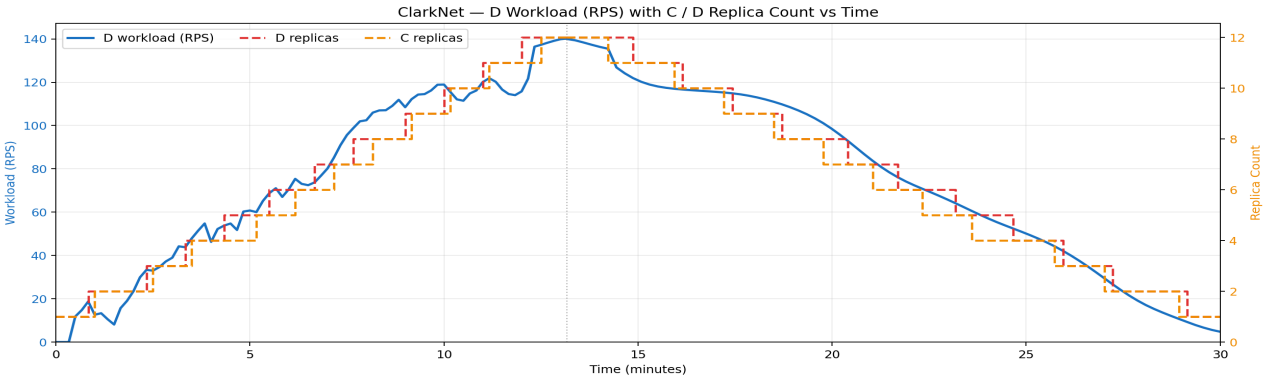


Figure 11: Container replica count with arrival rate for Reactive (C) and Proactive (D) — Periodic Workload

8.6 Comparative Discussion

SLA Compliance and Latency. Across all four workloads, Exp D (proactive) consistently achieves higher SLA compliance and lower P95 latency than Exp C (reactive). The most pronounced improvement occurs on the periodic workload (-43.3% P95 latency), where the ensemble predictor accurately anticipates each rising half-cycle and pre-scales replicas $\approx$60 seconds before the reactive CPU threshold is crossed. The NASA workload shows a 40.5% latency reduction, reflecting the predictor's ability to handle bursty, irregular demand. The ClarkNet workload yields a 28.6% latency reduction despite high-frequency oscillations. The increasing workload shows the smallest gain (6.5%), consistent with both controllers being driven by the ramp trajectory rather than transient dynamics.

Resource Efficiency. A key finding is that proactive autoscaling does not systematically over-provision. The proactive autoscaler improves QoS with limited additional resource usage compared with the reactive autoscaler. Exp D uses 2.9% more average replicas on NASA, 2.6% fewer average replicas on ClarkNet, 2.0% more average replicas on the increasing workload, and 9.9% more average replicas on the periodic workload. Overall, the average replica count increases from 7.43 in Exp C to 7.65 in Exp D, which corresponds to only 2.9% average replica overhead. Thus, the proposed framework achieves better SLA compliance, lower latency, and fewer rejected requests without significant over-provisioning.

Request Rejection. On NASA, Exp D rejects only 5,663 requests versus 17,834 for Exp C, a 68.2% reduction. On ClarkNet, rejections fall by 47.8% (4,746 vs. 9,090). On the increasing workload, rejections decrease from 3,312 under Exp C to 2,985 under Exp D, corresponding to a 9.9% reduction. On the periodic workload, rejections decrease by 31.2% (6,599 vs. 9,590). These reductions directly translate into improved request handling and user-facing throughput.

CDF Analysis. The CDF plots confirm stochastic dominance of Exp D over Exp C across all latency percentiles in all four workloads. The separation is most visually striking on the periodic

workload, where Exp D's CDF reaches ≈0.90 by 400 ms while Exp C reaches only ≈0.84, a persistent 6-percentage-point gap across every oscillation cycle.

9 Achievements

1. **Objective 1 achieved:** The clustering-enhanced hybrid ensemble model (K-means + Prophet + LSTM + LR) was developed and evaluated against baseline prediction models. The proposed model achieved lower prediction error on the evaluated workload traces, demonstrating its ability to handle heterogeneous workload patterns more robustly.

2. **Objective 2 achieved:** The MAPE-K proactive autoscaling framework with predictive RPS and stability mechanisms eliminates reactive scaling lag, reducing mean P95 latency by 6.5–43.3% and SLA violations by 3.1–4.1 percentage points across all workload patterns.

3. **Objective 3 achieved:** A fully operational three-node cluster testbed with Docker Swarm, Prometheus, cAdvisor, and trace-driven Locust workload generation was implemented and validated.

4. **Objective 4 achieved:** Systematic comparison of four provisioning strategies No scaling, static replica, reactive auto-Scaling and proactive auto-scaling across four different workload patterns (NASA, ClarkNet, increasing, and periodic) provide comprehensive empirical benchmarking of containerized auto-scaling on Docker Swarm.

10 Conclusion

This research presents a proactive autoscaling framework for containerised cloud applications that addresses the fundamental limitation of reactive systems: scaling lag. By combining a clustering-enhanced hybrid ensemble workload predictor (K-means + Prophet + LSTM + Linear Regression) with a MAPE-K control loop on Docker Swarm, the proposed system anticipates demand changes in advance and provisions container replicas before workload surges materialize.

Experimental evaluation on a Docker Swarm cluster using four workload patterns demonstrates consistent improvements over reactive autoscaling: SLA compliance improves by 3.1–8.8 percentage points, mean P95 latency decreases by 6.5–43.3%, and request rejection rates fall by 9.9–68.2%. Overall, the proposed proactive autoscaling method uses only slightly higher infrastructure capacity while significantly outperforming the baseline autoscaling method in improving SLA compliance and reducing latency.

Future Work. Future directions include: (i) online model retraining to adapt to concept drift in long-running deployments; (ii) extension to multi-service microservice architectures with inter-service dependency modelling; (iii) vertical autoscaling integration (CPU/memory limit adjustment) alongside horizontal replica scaling; (iv) reinforcement learning-based policy optimisation for cost-SLA trade-off; and (v) evaluation on Kubernetes with HPA/VPA for broader platform applicability.

11 Publications

Published / Accepted Papers

1. Trivedi, N. S., & Upadhyaya, A. N. (2026). "Hybrid Ensemble Approach for Accurate Workload Prediction in Dynamic Cloud Environments." " *SSRG International Journal of Electronics and Communication Engineering*, 13(2), 180–194. https://doi.org/10.14445/23488549/IJECE-V13I2P114

2. Trivedi, N. S., & Panchal , S. D. (2023). "Optimization enabled elastic scaling in cloud based on predicted load for resource management." *Multiagent and Grid Systems*, 19(4), 289–311. https://doi.org/10.3233/MGS-230003

3. Trivedi, N. S., & Upadhyaya, A. N. "Proactive Autoscaling of Containerized Applications Using Hybrid Workload Prediction." (under review)

References

[1] Rodrigo N. Calheiros et al. "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS". In: *IEEE Transactions on Cloud Computing* 3.4 (2015), pp. 449–458. DOI: 10.1109/TCC.2014.2350475.

[2] Thanh-Tung Nguyen et al. "Horizontal Pod Autoscaler in Kubernetes for Elastic Container Orchestration". In: *IEEE Access* 8 (2020), pp. 219956–219971. DOI: 10.1109/ACCESS.2020.3044209.

[3] Q. Huang, S. Wang, and Z. Ding. "Autoscaling Method for Docker Swarm Towards Bursty Workload". In: *Computing and Informatics* 42.5 (2024), pp. 1037–1059. DOI: 10.31577/cai_2023_5_1037.

[4] Waheed Iqbal et al. "Dynamic Workload Patterns Prediction for Proactive Auto-scaling of Web Applications". In: *Journal of Network and Computer Applications* (2018). DOI: 10.1016/j.jnca.2018.09.023.

[5] Nguyen-Minh Dang-Quang and Myounghoon Yoo. "Deep Learning-Based Autoscaling Using Bidirectional Long Short-Term Memory for Kubernetes". In: *Applied Sciences* 11.9 (2021), p. 3835. DOI: 10.3390/app11093835.

[6] Shutian Luo et al. "The Power of Prediction: Microservice Auto Scaling via Workload Learning". In: *Proceedings of the 13th Symposium on Cloud Computing (SoCC '22)*. New York, NY, USA: ACM, 2022, pp. 355–369. DOI: 10.1145/3542929.3563477.

[7] Muhammad Abdullah et al. "Burst-Aware Predictive Autoscaling for Containerized Microservices". In: *IEEE Transactions on Services Computing* 15.3 (May 2022), pp. 1448–1460. DOI: 10.1109/TSC.2020.2995937.

[8] Federico Lombardi et al. "PASCAL: An Architecture for Proactive Auto-Scaling of Distributed Services". In: *Future Generation Computer Systems* 98 (2019), pp. 342–361. DOI: 10.1016/j.future.2019.03.003.

[9] Mohammad Abdullah, Waheed Iqbal, and Abdelkarim Erradi. "Predictive Autoscaling of Microservices Hosted in Fog Microdata Center". In: *IEEE Access* 9 (2021), pp. 112916–112930. DOI: 10.1109/ACCESS.2021.3103895.

[10] Hussain Ahmad et al. "Towards Resource-Efficient Reactive and Proactive Auto-Scaling for Microservice Architectures". In: *Journal of Systems and Software* (2025), p. 112390. DOI: 10.1016/j.jss.2025.112390.

[11] Renato Lopes de Souza, Tiago Ferreto, and César A. F. De Rose. "Predicting and Avoiding SLA Violations of Containerized Applications Using Machine Learning and Elasticity". In: *Journal of Network and Computer Applications* 197 (2022), p. 103268. DOI: 10.1016/j.jnca.2021.103268.

[12] Javad Dogani, Farshad Khunjush, and M. Reza Mahmoudi. "Proactive Auto-Scaling for Web Applications in Container-Based Edge Computing Using Federated Learning". In: *Journal of Systems Architecture* 147 (2024), p. 103063. DOI: 10.1016/j.sysarc.2023.103063.

[13] A. Lipari, G. P. Mattia, and R. Beraldi. "Dynamic and Forecast-Based Containers Autoscaling for Kubernetes with Reinforcement Learning". In: *Proceedings of the 2025 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. Milano, Italy: IEEE, 2025, pp. 1081–1088. DOI: 10.1109/IPDPSW66978.2025.00169.

[14] Chamath Wanigasooriya and Indrajith Ekanayake. "NimbusGuard: A Novel Framework for Proactive Kubernetes Autoscaling Using Deep Q-Networks". In: *Proceedings of the 40th International Conference on Information Networking (ICOIN)*. IEEE, 2026, pp. 726–731. DOI: 10.48550/arXiv.2604.11017.

[15] B. Kumar, A. Verma, P. Verma, et al. "Optimizing Resource Allocation in Cloud-Native Applications Through Proactive Autoscaling with the InformerAutoScale Model". In: *The Journal of Supercomputing* 81 (2025), p. 1077. DOI: 10.1007/s11227-025-07500-7.

[16] B. Kumar, A. Verma, and P. Verma. "A Multivariate Transformer-Based Monitor-Analyze-Plan-Execute (MAPE) Autoscaling Framework for Dynamic Resource Allocation in Cloud Environment". In: *Computing* 107 (2025), p. 69. DOI: 10.1007/s00607-025-01426-x.

[17] Mouhamadou Bâ, Chandra Harsha, and Hari Subramanian. "Leveraging Interpretability in the Transformer to Automate Proactive Scaling of Cloud Resources". In: *IEEE Transactions on Cloud Computing* (2024). DOI: 10.1109/TCC.2024.3345678.

[18] J. D. Freitas et al. "Evaluating Window Size Effects on Univariate Time Series Forecasting with Machine Learning". In: *Journal of Information and Data Management* 16.1 (2025), pp. 213–223. DOI: 10.5753/jidm.2025.4668.

[19] R. S. Shariffdeen et al. "Workload and Resource Aware Proactive Auto-Scaler for PaaS Cloud". In: *Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. 2016, pp. 11–18. DOI: 10.1109/CLOUD.2016.0012.

[20] Hoang Minh Nguyen et al. "ESNemble: an Echo State Network-based ensemble for workload prediction and resource allocation of Web applications in the cloud". In: *The Journal of Supercomputing* 75 (2019), pp. 6303–6323. DOI: 10.1007/s11227-019-02851-4.

[21] Deepika Saxena et al. "Performance Analysis of Machine Learning Centered Workload Prediction Models for Cloud". In: *IEEE Transactions on Parallel and Distributed Systems* 34.4 (2023), pp. 1313–1330. DOI: 10.1109/TPDS.2023.3240567.

[22] Parminder Singh, Pooja Gupta, and Kiran Jyoti. "TASM: Technocrat ARIMA and SVR Model for Workload Prediction of Web Applications in Cloud". In: *Cluster Computing* 22.2 (2019), pp. 619–633. DOI: 10.1007/s10586-018-2868-6.

[23] S. Thiruchadai Pandeeswari et al. "Predictive Cloud Resource Management by Workload Forecasting with an ensemble of Hybrid ARIMA-LSTM and EL models". In: *Anadolu Psikiyatri Dergisi* 26.2 (2025). DOI: 10.5281/zenodo.15038432.

[24] P. B. Guruge and Y. H. P. P. Priyadarshana. "Time Series Forecasting-Based Kubernetes Autoscaling Using Facebook Prophet and Long Short-Term Memory". In: *Frontiers in Computer Science* 7 (2025), p. 1509165. DOI: 10.3389/fcomp.2025.1509165.

[25] Liang Bao et al. "On accurate prediction of cloud workloads with adaptive pattern mining". In: *Journal of Supercomputing* 79.1 (2023), pp. 160–187. DOI: 10.1007/s11227-022-04647-5.

[26] "Adaptive Workload Prediction for Cloud Systems". In: *Proceedings of the IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2024. DOI: 10.1109/SOSE62363.2024.00011.

[27] M. Nashaat et al. "Dynamic machine learning approach for workload prediction in cloud environments". In: *Scientific Reports* 16.1 (2026), p. 10983. DOI: 10.1038/s41598-026-40777-z.