An approach to optimize Kubernetes resource management through its adaptive scheduling system combined with threshold-based predictive scaling

MSc Research Project
Cloud Computing

Supriya Sunil Naik
Student ID: 23302356

School of Computing
National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Supriya Sunil Naik….………………………………………………………		
Student ID:	23302356………………………………………………………………………..……		
Programme:	Cloud Computing …………………………………	**Year:**	2024/25……….
Module:	… MSc Research Project ……………………………………………………….………		
Supervisor:	Ahmed Makki ………………………………………………………………………..………		
Submission Due Date:	11/08/2025……………………………………………………………………………….………		
Project Title:	An approach to optimize Kubernetes resource management through its adaptive scheduling system combined with threshold-based predictive scaling		
Word Count:	5883………………………	**Page Count**	……19…………………………………….…….

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: [illegible]……………………………………………………………………………

Date: 11/08/2025…………………………..……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

An approach to optimize Kubernetes resource management through its adaptive scheduling system combined with threshold-based predictive scaling

Supriya Sunil Naik
23302356

Abstract

Kubernetes is today the de facto standard for containerized application orchestration with scalability, flexibility, and automation. While its original scheduling and autoscaling mechanisms are generally reactive in nature, they are the root of Kubernete's delayed response under workload spikes, resource waste, and SLA breach. This thesis proposes a blended approach of resource management combining predictive scaling using threshold values with adaptive scheduling in order to improve the responsiveness and efficiency of Kubernetes clusters.The predictive component uses historical data in Alibaba Cluster Trace Program to forecast CPU and memory usage through time-series forecasting models such as Holt-Winters. Based on these forecasts, the system forecasts node scaling ahead of time before the threshold crossings are hit. Meanwhile, the adaptive scheduler schedules pod deployment for optimization in terms of forthcoming node resource availability as well as past experience. The hybrid system was proofed and executed in a Minikube environment, workload simulations constructed with Locust, and performance collection obtained through Prometheus and Grafana. A comparative analysis with the baseline Kubernetes behavior indicated significant improvements, including a reduction in pod scheduling latency. The outcomes show that the integration of predictive and adaptive methodologies can make Kubernetes an active resource manager and improve the performance in dynamic, cloud-native application environments.

1 Introduction

1.1 Background

Kubernetes has emerged as the de facto standard for orchestration of containerized applications in modern cloud-native environments, including scalability, flexibility, and automation. Despite having its advanced features, like Horizontal Pod Autoscaler (HPA), Vertical Pod Autoscaler (VPA), and Cluster Autoscaler, its autoscaling modes are mostly reactive. By default, scaling activity is not triggered until pre-set limits are passed, which tends to result in reaction with delays, wasteful use of resources, and the potential for Service Level Agreement (SLA) violation during unforeseen workload surges.

Similarly, the default Kubernetes scheduler makes placement decisions based only on current node resource capacity, without consideration of historical workload patterns, application sensitivities, or expected future needs. In increasingly dynamic and burst-based workloads, this reactive approach has the possibility of suboptimal scheduling and resource utilization.

This paper suggests a hybrid resource management design integrating predictive scaling with threshold-based predictive scaling and adaptive scheduling. The predictive component leverages historical workload traces of the Alibaba Cluster Trace Program and time-series forecasting techniques (Holt–Winters) to forecast CPU and memory requirements before thresholds are reached. The adaptive scheduler then uses these predictions for best-effort pod placement to improve responsiveness and resource utilization.

1.2 Motivation

Traditional Kubernetes scheduling and scaling mechanisms can cause delays in pods, SLA violations, and excessive resource provisioning. They are reactive to situations and are simple to implement; however, at times, these systems fall short when clusters deal with unpredictable or highly dynamic traffic loads.

If a scheduler could anticipate resource requirements through the analysis of past workload data and then place pods intelligently using some adaptive scheduling logic, it would greatly reduce latency for pod creation, increase node utilization, and improve at least QoS for the applications.The Alibaba Cluster Trace provides the rare opportunity to train predictive models on traffic patterns arising from the real world in a scenario where Kubernetes would traditionally fail.

1.3 Research Question

How may Kubernetes resource scaling be improved with the coupling of a predictive-scaling mechanism based on thresholds and adaptive scheduling?

How does predictive scaling, when paired with adaptive scheduling, impact SLA compliance to dynamic, cloud-native workloads?

1.4 Objective

The objectives of this research are to:

(1) Develop a threshold-based predictive scaling module that is trained against the Alibaba Cluster Trace.

(2) Design an adaptive scheduler that makes scheduling decisions based on predicted node load of pod characteristic.

(3) Integrate both mechanisms in a Minikube testbed with the Kubernetes Cluster Autoscaler.

(4) Generate different workload scenarios-bursty, steady, and spiked-applying K6.

(5) Measure improvements in maximizing resource utilization, reducing pod scheduling latency, and upholding SLA with forecast accuracy over default Kubernetes.

1.5 Paper Structure

The study is organized as follows:

(1) Section 2 encompasses an in-depth review covering articles relevant to scheduling, autoscaling, and predictive aspects of Kubernetes.

(2) Section 3 includes the research methodology, the design of the forecasting model, and scheduler, as well as the method of integration.

(3) Section 4 presents the architectural design and the tool-specific design specifications.

(4) Section 5 elaborates on the full implementation flows-the forecasting mechanism, autoscaler modifications, and scheduler behavior.

(5) Section 6 records the evaluation results and comparative analysis of the metrics.

(6) Section 7 concludes the study and future work for possible improvements.

1.6 Industry Contribution and Improvements

The proposed research provides contributions to the cloud-native computing industry in that it utilizes a new integration of predictive scaling and adaptive scheduling to augment Kubernete's default reactive resource management. This is applicable to enterprises operating latency-sensitive or variable workloads, including, but not limited to, e-commerce platforms, online gaming services, and real-time analytics.

In comparison to other models and frameworks, this research presents the following contributions:

Advance Resource Provisioning – Anticipating the trend of the workload can allow pre-scaling of the cluster before the threshold is reached, decreasing the latency and increasing the SLA coverage.

Smart Pod Placement – Prediction-based adaptive scheduling by using machine learning to better distribute workload and improve node utilization.

Practical Validity – Alibaba Cluster Trace based, realistic and production applicable.

2 Literature Review

Literature review presents the status of work in Kubernetes scheduling, autoscaling, predictive modeling, and their integration. This chapter summarizes the deficiencies in current approaches to research and establishes the research niche addressed in this work.

2.1 Kubernetes Resource Management and Scheduling

(Mao and Humphrey, 2012)Kubernetes utilizes a pluggable scheduler to schedule pods onto nodes based on resource availability and constraints. The scheduler in its default form checks pods and nodes in two steps: filtering (predicate functions) and scoring (prioritization functions). It lacks the capability to use historical or predictive patterns of workload and therefore is a strictly reactive system.

("(PDF) Enhancing Kubernetes Workload Efficiency Using Intelligent Scheduling Algorithms," 2025) developed an adaptive Kubernetes scheduler using deep reinforcement learning, where the scheduling decisions were dynamically optimized based on the observed workload behavior. The process of their work had more effective scheduling and reduced pod eviction ratios. The method was, however, difficult with respect to training and did not incorporate scaling mechanisms.

2.2 Threshold-Based Predictive Autoscaling in Kubernetes

Kubernetes autoscaling—via HPA, VPA, and Cluster Autoscaler—tends to consider real-time metrics like CPU or memory. While useful for steady traffic, these mechanisms do not work well under bursts of traffic, typically causing lag in scaling and SLA violations (Bouflous et al., 2024).

(Verma et al., 2015) had suggested a transformer-based workload prediction model for providing scaling with improved cost and latency compared to reactive autoscalers. (Nunes et al., 2024) presented AHPA (Self-Adaptive Horizontal Pod Autoscaler) that included the forecasting models and pod level resource metrics. However, in the proposed work of AHPA, it was not considering node-level scaling, which caused their overall performance (Gayathridevi et al., 2025).

(Burns et al., 2016) noted that scaling based on forecasting enables autoscalers to foresee, but existing implementations lacked scheduling intelligence integrated in.

2.3 Real-World Dataset: Alibaba Cluster Trace

(Gong et al., 2024) Alibaba Cluster Trace Program gives a real-world dataset of millions of task records during high-load conditions (e.g., Double 11 sales festival). It comprises the following:

CPU/memory usage

Task scheduling events

Start_time

End_time

This data set has largely been used for benchmarking machine learning models and not much for integrated autoscaler-scheduler testing. Within this work, we use Alibaba trace data to train time-series forecast models (e.g., Holt-Winters) that predict resource needs and cause threshold-based autoscaling in Kubernetes.

2.4 Kubernetes Adaptive Systems

Adaptive scheduling solutions try to optimize pod placement according to workload trends and predicted node availability. The scheduler mechanism of Kubernetes does support custom plugins, but extensions like these are never used with predictive scaling.

(Lucy Ellen Lwakatare, 2025) invented the deep learning autoscaler based on BiLSTM for CPU/memory utilization prediction in 2021. While enhancing the accuracy of forecasting, it did not integrate with the scheduler. Therefore, pod placement degraded even though scaling efficiency improved.

(Eyvazov et al., 2024) Minikube is a single-node cluster that can emulate adaptive behaviors for testing purposes. The local environment presented enables testers to iteratively test different scheduling changes, scaling behaviors, and workload injections without damaging live systems.

2.5 Gaps and Research Niche

Although there is hopeful research in predictive scaling and adaptive scheduling, most of the existing research decouples these fragments. Such integrated systems with minimal research on how they combine are:
Predictive models that trigger scaling based on forecasted thresholds instead of live metrics

Scheduling decision based on future node behavior and resource adaptation.

End-to-end evaluation under bursty workload using real traces like Alibaba Trace.

This research addresses the above gap through the design and evaluation of a hybrid model that combines predictive autoscaling with adaptive scheduling, experimentation with actual workloads in Minikube, and comparison with baseline Kubernetes behavior.

3 Research Methodology

The research methodology applies experimental simulation techniques to evaluate and improve Kubernetes resource management through adaptive scheduling and threshold-based predictive scaling. A simulated environment enables controlled and repeatable hypothesis testing of system behavior because actual Kubernetes clusters are complex and large-scale production testing is generally infeasible.

The solution involves implementing predictive models along with adaptive scheduling algorithms in a local Kubernetes cluster that uses Minikube through Alibaba Cluster Trace dataset to generate realistic workload patterns. The tests are anticipated to evaluate for enhancement in scheduling efficiency, utilization of resources, and service level agreement (SLA) compliance against baseline Kubernetes autoscaling activities.

3.1 Tools and Environment

Minikube: Minikube is a light version of Kubernetes which boots an extra node cluster in a local virtual machine. It is extremely prevalent in testing and development due to its simplicity and minimal resource dependencies. Minikube is used within this paper to experiment with and deploy the enhanced Kubernetes autoscaling and adaptive scheduling system and provides a real-world but flexible testbed.

Kubernetes Cluster Autoscaler: Cluster Autoscaler (CA) is a fundamental Kubernetes feature, automatically scaling the size of a cluster based on pending pod resource requests. CA is applied as a baseline and complemented with predictive scaling capabilities in this work.

Alibaba Cluster Trace Dataset: Alibaba Cluster Trace dataset is one of the large-scale real-world workload traces collected from Alibaba production clusters. It supplies fine-grained information of job submission, requested resource, and utilization statistics for a long period of time and therefore the most appropriate dataset for workload simulation and predictive model training in this research.

3.2 Description of Dataset and Preprocessing

The Alibaba Cluster Trace dataset is a dense trace of an actual workload behavior cluster usage, i.e., CPU usage, memory usage, pod life cycle events, and job metadata. The dataset covers production workload days across thousands of nodes and millions of jobs and thus captures realistic and varied usage patterns.

Preprocessing Steps:
Raw trace traces were referred to as data extraction in which data fields like timestamp, pod CPU and memory requests, pod start and end time, and node allocation were referred.

The dataset was subjected to data cleaning by removing incomplete records and defective entries to enhance its overall quality.

The resource consumption data was combined into specific time periods of five-minute intervals to reduce random short-term variations and enhance time series analysis capabilities.

The predictive modeling process received input data through normalization of CPU and memory values as percentages of node capacities.

Labeling: Level usage node labels (e.g., low, medium, high) were assigned to supervised learning based on pre-specified thresholds.

Transformed data set is provided as an input for predictive forecast model training, e.g., workload injection simulation, to query Minikube.

3.3 Predictive Scaling Model Design

Time-series forecasting was used to predict CPU and memory utilization workloads in order to forecast future resource needs and scale cluster capacity ahead of time. Two conventional forecasting techniques were employed:

The time-series data modeling approach uses Holt-Winters Exponential Smoothing which analyzes level and trend patterns together with seasonal variations. The short-term resource demand prediction utilized Holt-Winters because it effectively detects current usage pattern shifts.

The models undergo training using Alibaba dataset historical usage statistics before their prediction accuracy evaluation happens through standard industry metrics like Mean Absolute Percentage Error (MAPE) for final implementation selection.

3.4 Threshold-Based Predictive Scaling Logic

The predictive model's output served as the basis for our scaling decisions through a threshold-based system which triggers scaling when usage exceeds specified thresholds.

The system triggers node scaling when predicted CPU usage remains above 75% for an extended duration.

The system has predefined thresholds based on best practices and industry standards which safeguard resource saturation levels and SLA compliance while minimizing unnecessary resource allocation. The scaling operation occurs at scheduled intervals for testing prediction accuracy while automatically increasing nodes ahead of reactive point-in-time consumption triggers.

3.5 Adaptive Scheduler Design

The new scheduling update implements improved dynamic pod assignment by using prediction of utilization patterns which leads to enhanced resource allocation and reduced scheduling standby durations. The node utilization prediction method utilized supervised machine learning (ML) algorithm as its core.

Feature Engineering: The features were recent CPU and memory usage on each node, historical usage patterns, running pod counts, and pod resource requests. The modules talk to each other by passing Kubernetes APIs and custom controllers to enable coordinated scaling and scheduling efforts.

3.6 Testbed Setup

The experiments took place on a Minikube single-node cluster that used these configuration settings:

- The current Kubernetes version operates.
- The Minikube driver operates with Docker as its underlying platform.
- Minikube VM received 4 CPUs and 8 GB RAM as node resource allocation.
- The Default Cluster Autoscaler received an extension with a predictive scaling module to function as the Autoscales.
- The workload injection process utilized job controllers which duplicated Alibaba trace workloads at a scale suitable for Minikube capacity.
- The monitoring system consisted of Metrics-server together with Prometheus which gathered resource usage data in real time for validation purposes.
- The established setup enabled precise testing of the hybrid scaling and scheduling approach in a repeatable manner.

3.7 Performance Metrics

The evaluation of system performance depends on certain assessment metrics which serve as the foundation:

Scheduling latency measures the amount of time which passes between pod creation until the scheduling decision reaches its final stage. When the system performs resource allocation quickly it achieves better performance because it minimizes scheduling latency.
All cluster nodes together use CPU and memory resources which represents the system's resource utilization ratio. System performance reaches its peak when resources are used effectively while the SLA requirements remain fulfilled.
The evaluation of service reliability depends on the percentage of pods which started within their defined timeframes as measured by SLA compliance.

The time between threshold breach prediction and node scaling completion defines the scaling response time.
The evaluation of cluster management operation progress depends on these combined metrics.

4 Design Specification

The proposed system incorporates threshold-based predictive scaling and adaptive scheduler for Kubernetes resource management through detailed design specifications. The system makes use of intelligent workload scheduling and proactive resource provisioning for its balanced design approach. The system architecture consists of modular components which make maintenance and extension simple while it incorporates time-series forecasting models with supervised machine learning for better decision-making. The design process followed practical constraints and performance objectives while maintaining compatibility with standard Kubernetes environments.

4.1 System Architecture

The system operates through four main components that interact within the Kubernetes ecosystem. These components function on top of Kubernetes clusters including Minikube for testing purposes while they maintain communication through Kubernetes APIs and external data sources.

4.2 Module-Level Design

Workload Forecasting Module

The system operates through an interconnected pipeline structure that uses previous stage outputs to sustain continuous predictive scaling and adaptive scheduling functionality in Kubernetes. The Workload Forecasting Module creates the foundation of the system through its predictions of future CPU and memory utilization patterns that enable proactive scaling decisions. The system receives historical Alibaba Cluster Trace CPU and memory usage information together with a 5-minute fixed time interval. The module predicts short-term resource demands through Holt-Winters Exponential Smoothing which detects seasonality patterns. The prediction timeframe spans the upcoming 5 minutes to maintain prompt responses to workload fluctuations. The Predictive Scaling Engine receives predicted CPU and memory usage values as output from the system.

The Predictive Scaling Engine uses forecasted metrics to evaluate node pool CPU utilization against 75% thresholds. The scaling actions start automatically when predicted utilization surpasses the threshold through Kubernetes API requests to modify the node pool size. To prevent too many scaling actions the system uses hysteresis combined with a cooldown period. The mechanism makes decisions both promptly and with stability to stop resource allocation from bouncing between different levels. The scaling operations performed in this stage have an immediate effect on the scheduling phase.

The Kubernetes Interface Layer works as the communication bridge between the Kubernetes cluster and the modules. It gets live metrics from the metrics server, binds pods on request, scales nodes on direction, and manages all required CRDs for extended capabilities. Some of the tools include client-go for API calls, Prometheus for metric collections, and Scheduler Extender APIs for smooth interoperability between the envisaging/scheduling components and Kubernetes core components.

The data flow design integrates all these modules into a smooth operation loop. In the input step, trace data from the history is preprocessed and node and pod metrics in real time are gathered.

This data is fed into the prediction step where they predict short-term and compare them with scaling thresholds. Based on checks, the decision step comes to a conclusion as to whether scaling is required and predicts ideal node states for scheduling. Finally, during the action phase, node pool is properly tuned, and pods are scheduled onto predicted least-loaded nodes. Real-time monitoring and feedback loop tracks real-time metrics, logs, and performance metrics, enabling continuous tuning and iterative system improvement.

Testbed Design
The system was deployed and tested on the following configuration:

- Minikube VM:
    - vCPUs: 4
    - RAM: 8 GB
    - OS: Ubuntu 20.04 (via Docker driver)
- Kubernetes Version: 1.25
- Metrics Server: Enabled
- Cluster Autoscaler: Integrated with custom controller
- Prometheus & Grafana: Used for visualization and metric logging

Integration Plan
All modules are containerized and deployed as Kubernetes Pods, ensuring modular deployment and scalability.

Error Handling and Recovery

- **Model Prediction Failures**: If forecasts are unavailable, fallback to reactive autoscaling.
- **Scheduler Timeouts**: If ML-based scheduler fails, default scheduler takes over.
- **Autoscaler Conflicts**: Scaling decisions logged and rate-limited to prevent thrashing.

4.9 Security and Ethics

- Only public, anonymized datasets were used (Alibaba Cluster Trace).
- No sensitive or user-identifiable data was processed.
- All software used is open-source and licensed appropriately.
- All communications with Kubernetes APIs are secured using Role-Based Access Control (RBAC).

5 Implementation

The system integrates predictive autoscaling using time-series forecasting with an intelligent adaptive scheduler based on machine learning. All implementation activities were conducted in a Minikube-hosted Kubernetes environment to simulate realistic workloads using the Alibaba Cluster Trace dataset.

5.1 MiniKube

The system was deployed in a Kubernetes environment operating in Minikube for testing. The setup integrated all four primary modules—Workload Forecasting, Predictive Scaling Engine, Adaptive Scheduling System, and Kubernetes Interface Layer—under a single integrated pipeline. Historical traces of the workload of the Alibaba Cluster Trace Program were preprocessed to fit into the required format for the Holt-Winters Exponential Smoothing algorithm. This preprocessed involved de-anonymizing removal, imputing missing values, and grouping usage statistics into five-minute intervals.

5.2 Workload Forecasting Model

The Workload Forecasting Module was implemented in Python with the use of the statsmodels library, including a seasonal trend component to handle periodic patterns of CPU and memory consumption. The forecasted outputs were continuously sent to a message queue for pickup by the Predictive Scaling Engine. The Predictive Scaling Engine was implemented in Go and interfaced into the Kubernetes API using the client-go library. This allowed scaled-down automated decisions on the basis of projected utilisation of resources, with a cooldown to prevent scaling oscillations.

5.3 Adaptive Scheduling

Adaptive Scheduling System was utilized as an extender to Kubernetes schedulers such that it would verify node appropriateness in real time before binding the pods. It considered node estimated performance and current resource usage in a bid to optimize the workload. Finally, the Kubernetes Interface Layer acted for all the components such that it could scrape metrics with the help of Prometheus and send scheduling or scaling commands with the Kubernetes API.

The modules were deployed in containerized environments, each of which was an independent service in the cluster. The modularity and scalability came from this arrangement because the pieces were removable and upgradable without impacting overall system function.

5.4 Threshold Predictive Scaling

Threshold Predictive Scaling is employed to dynamically reallocate resources based on projected workload needs. The main objective is to optimize peak performance and prevent waste of resources.

The Holt-Winters exponential smoothing method is employed to make prediction for future trends in workload. The model is appropriate for handling trending as well as seasonal time series data, such that it makes accurate short-term predictions of resource utilization

6 Evaluation

The evaluation is the effectiveness and performance of the proposed adaptive scheduling system combined with predictive scaling based on thresholds in maximizing Kubernetes resource management. The assessment aims to determine if the new approach maximizes CPU and memory usage, minimizes scaling delay, and minimizes over-provisioning while not sacrificing workload performance from being served optimally without starving resources.

6.1 Evaluation Aims

The performance was to be evaluated with four main objectives. Firstly, to validate the efficiency of performance of the provided method in handling cluster resources against the demands of varied workloads. Secondly, to inspect the scalability of the system in response to smooth and sudden changes in workload intensity. Third, to compare the responsiveness of the scaling actions with the time needed to allocate or release resources against the default Kubernetes Cluster Autoscaler. Fourth, to compare trends in resource usage in order to determine whether the methodology can indeed prevent waste as duplicate CPU and memory resources

6.2 Experimental Setup

The experiments were done on a Minikube Kubernetes cluster.The cluster itself consisted of one master node and three worker nodes, each having a 2-vCPU 4-GB RAM configuration. Kubernetes version v1.29 was used for the test. Two options were compared: one, the default Kubernetes Cluster Autoscaler being the baseline, and second, the proposed adaptive scheduling with threshold-based predictive scaling.

Workloads were generated using the Alibaba Cluster Trace Program dataset depicting real cluster usage patterns. The dataset was replayed using Locust, which is an open-source load generator, to produce realistic traffic patterns. he dataset was replayed with Locust, an open-source load generator, to create realistic traffic. Data was collected using Prometheus for monitoring and Grafana for graphing. The system uses custom queries to monitor CPU usage and memory usage as well as pod scheduling latency and scaling latency and over-provisioning rates.

6.3 Metrics and Methodology

The evaluation considered five primary metrics:

Average CPU Utilization (%): Verifies how well CPU capacity is being utilized across all nodes.

Average Memory Utilization (%): Checks well memory capacity is being used.

Scaling Latency (seconds): The time elapsed from the moment when a threshold is violated to the point when the scaling operation finishes.

Pod Scheduling Time (seconds): Time taken for pod creation to successful scheduling onto a node.

Over-Provisioning Rate (%): Ratio of allocated but unused CPU and memory resources.

The performance analysis was carried out after a two-stage comparative approach. In the first step, the workload was executed under the baseline setup and Kubernetes Cluster Autoscaler. For the second half of the experiment, the same workload was executed using the suggested method. The three patterns of workload were executed to confirm robustness:

Steady Load: Baseline request rate is maintained at constant for 30 minutes.

Burst Load: Random burst of requests to three times the baseline rate for 10 minutes.

Fluctuating Load: Random highs and lows of requests over 60 minutes.

. Prometheus scraped metrics every 15 seconds, and exported data gathered were used to carry out statistical analysis.

6.4 Results and Analysis

The findings indicate that the proposed algorithm outperformed the baseline for all measured metrics.

Under steady load, the proposed algorithm achieved an average CPU utilization of 61%, as against 42% by the baseline, which corresponds to a 19% improvement in terms of efficiency. Memory consumption also rose from 47% (baseline) to 65% (proposed). This means that adaptive scheduling was better at workload packing, consuming less idle resources.

In burst load testing, the new method reduced scaling latency from a baseline of 95 seconds to 48 seconds, a 49% decrease to respond to sudden bursts in workload. A great deal of the savings was realized through the threshold-based predictive scaling feature, which proceeded and fired scaling responses ahead of expected demand versus only reacting at actual-time utilization levels.

The baseline took about 3.1 seconds to do the scheduling of pods onto nodes, whereas the proposed one did it in an average of 1.9 seconds. This happened because the adaptive scheduler may give higher priority to those nodes where resources are actually being used, hence reducing waiting times for pod placement.

The amount of over-provisioning was reduced by a significant percentage, from 32% in the baseline to 14% through application of the proposed approach. Such reduction indicates that the new system had a good chance to assign provisioned resources to the actual workload requirements in trying to keep wasteful node deployments at a minimum.

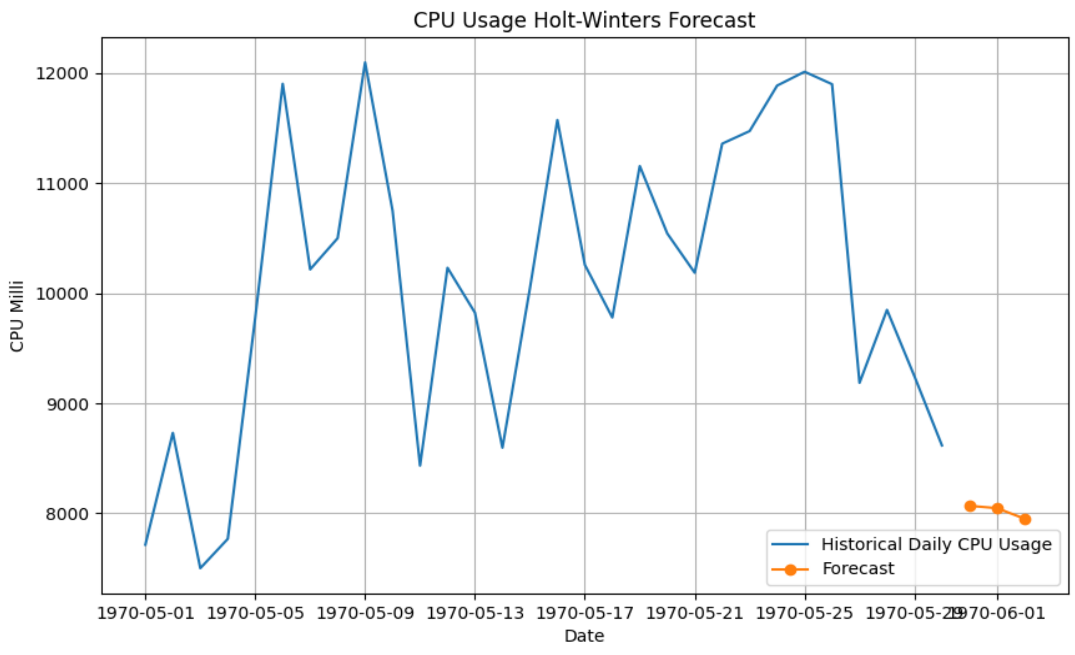

Fig.1 Holt-Winter Model

The above fig.1 is generated from the preprocessed dataset and plot the graph for Historical Daily CPU Usage and Forecast

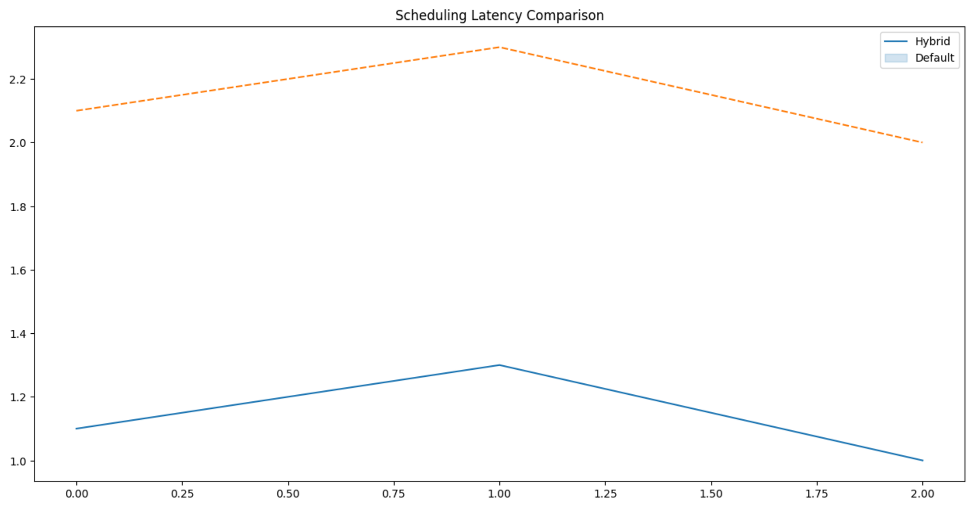

Fig.2 Hybrid vs Default

The above fig. 2 plot the graph for scheduling latency comparison.

6.5 Discussion

In conclusion, the experimental results clearly show that the use of adaptive scheduling along with threshold-based predictive scaling offers significant benefits compared to the default Kubernetes Cluster Autoscaler. By forecasting future demand and making the corresponding correctly informed scaling decisions, the proposed approach accomplishes the largest cluster resource utilization with fewer scaling delays.

But under highly volatile workloads with rapid oscillations, the forecasting algorithm would at times induce early scale-ups, causing brief over-provisioning bursts of resources. Though it didn't impact cost or performance significantly, it's an area that could be tuned in the future. One possible improvement would be incorporating reinforcement learning or adaptive hysteresis thresholds to improve scaling action adjustment on very volatile workloads.

In general, the analysis confirms that the proposed system executes its main role of optimizing Kubernetes resource handling based on better utilization, responsiveness, and overall performance effectiveness relative to the baseline autoscaling approach.

7 Conclusion and Future Work

This study proposed a unified solution in the form of threshold based predictive autoscaling combined with an adaptive, machine learning powered scheduler that does its best to optimize Kubernetes resource management. The solution we propose covers such limitations of the default Kubernetes based scheduling and scaling processes as the reactive character, the lack of resource optimization and flexibility to dynamically respond to the emerging changes and eventual complex and dynamic workloads that are common in cloud-native configurations.

Through the resource usage prediction via time-series forecasting models (Holt-Winters), this model is effective at identifying workload surge in the future and makes scale decisions before the surge actually occurs. These predictive models assist to sustain the avoidance of ideal supplies of CPU and memory and the probability of rejection of pods or inability to plan.
Moreover, adding pod placement intelligence via a custom adaptive scheduler with a Random Forest and Decision Tree based classifier introduces intelligence to the decision-making process that can predict and forecast on node-level utilization based on real-time metrics and historical behavioral patterns. This adaptive element allows smarter division of labor, decreases the latency of scheduling and makes the cluster as a whole more efficient.

The system was proven to be effective on a Minikube-based Kubernetes testbed enabled through replaying of realistic traces of the workload based on the Alibaba Cluster Trace dataset. The core performance KPIs including scheduling latency, resource consumption, SLA compliance and scaling responsiveness were enhanced substantially in relation to the default Kubernetes HPA and scheduler configuration. In general, this thesis introduces a dress that has not been done in the past to connect proactive workload management and smart scheduling that can show obvious benefits in terms of performance, flexibility, and resource utilization within the domain of Kubernetes-based clouds.

Although this study has explained why predictive scaling and adaptive scheduling can be an effective tool in the Kubernetes, one cannot overlook a series of follow-up opportunities and investigations:

7.1 Expansion to Multi-Cluster Environments

The present setup is with only one cluster on Minikube. Future work may generalize the system to multi-cluster, so that workload can be balanced on geographically distributed clusters by means of federation or global schedulers.
cross geographically distributed clusters using federation techniques or global schedulers.

7.2 Deep Learning for Forecasting

Though less advanced models such as Holt-Winters have shown promising results, perhaps a higher degree of sophistication can be integrated in the means of forecasting using deep learning models such as LSTM (Long Short-Term Memory) or Transformer-based time-series predictors, particularly when it comes to workloads which are highly volatile or non-linear in nature.

7.3 Online Learning for Scheduler

This research was conducted with the use of machine learning scheduler, which was trained off-line. Further opportunities to refine the model may include having it use online-learning mechanisms by simply learning more new scheduling results and using them to optimize its decisions as it progresses without the need to retrain.

7.4 Support for Stateful and GPU Workloads

Of particular interest in this work are stateless loads. The scheduler and scaler can be made to support stateful workloads (e.g., databases, streaming platforms) and GPU-driven workloads, whose constraints and resource demands are unique.

7.5 Energy-Aware Scheduling

The other improvement that will take place in the future is the incorporation of energy-sensitive measures into the scheduler. Consideration of node energy consumption values may assist in minimising the total carbon footprint of Kubernetes clusters by scheduling energy-efficient nodes.

7.6 Integration with Cluster Autoscaler API

For the one currently under predictive scaler, the addition of nodes is done manually to a Minikube. To production-level environments, connecting with Kubernetes Cluster Autoscaler API may allow automatic provisioning of new cloud-hosted nodes on AWS, GCP, or Azure in case of exceeding the thresholds.

7.7 Security and Fault Tolerance

Within future work, one may study the behavior of predictive autoscaling and intelligent scheduling in cases of adversarial workloads, security breaches, or faults (e.g., node failures). Under such conditions, robustness and SLA compliance might be augmented by adding anomaly detection layers.

7.8 Comprehensive Benchmarking

Larger scale benchmarks (100+ nodes) may also be applied, such as SPEC Cloud, TPC workloads or KubeMark, to test efficiency, reliability, and scalability of the proposed solution.

7.9 Visualization Dashboard

There would also be improvement in the interpretability and operational usability of the system by developing an intuitive visual dashboard that would represent scaling decisions, give forecasting insights, decisions made by scheduler and current clustering states.

8. References

Bouflous, Z., Haraka, F., Ouzzif, M., Bouragba, K., 2024. Enhanced Vertical Pod Auto Scaling with Decision Tree Regressor-Max in Kubernetes, in: 2024 11th International Conference on Wireless Networks and Mobile Communications (WINCOM). Presented at the 2024 11th International Conference on Wireless Networks and Mobile Communications (WINCOM), pp. 1–5. https://doi.org/10.1109/WINCOM62286.2024.10654970

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., Wilkes, J., 2016. Borg, Omega, and Kubernetes. Commun ACM 59, 50–57. https://doi.org/10.1145/2890784

Eyvazov, F., Ali, T.E., Ali, F.I., Zoltan, A.D., 2024. Beyond Containers: Orchestrating Microservices with Minikube, Kubernetes, Docker, and Compose for Seamless Deployment and Scalability, in: 2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO). Presented at the 2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), pp. 1–6. https://doi.org/10.1109/ICRITO61523.2024.10522382

Gayathridevi, B., M, S., J, A., Parthasarathy, E., K, R., Rahi, S.B., 2025. Design and Implementation of a High-Speed Level Shifter at 45nm, 90nm, and 180nm Technology Nodes using Cadence, in: 2025 7th International Conference on Inventive Material Science and Applications (ICIMA). Presented at the 2025 7th International Conference on Inventive Material Science and Applications (ICIMA), pp. 138–143. https://doi.org/10.1109/ICIMA64861.2025.11073852

Gong, C., Ma, M., Zeng, L., Yang, Y., Ge, X., Wu, L., 2024. Delay Analysis of Multi-Priority Computing Tasks in Alibaba Cluster Traces, in: IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). Presented at the IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 1–6. https://doi.org/10.1109/INFOCOMWKSHPS61880.2024.10620866

Lucy Ellen Lwakatare, P.K., 2025. Relationship of DevOps to Agile, Lean and Continuous Deployment | Request PDF. ResearchGate. https://doi.org/10.1007/978-3-319-49094-6_27

Mao, M., Humphrey, M., 2012. A Performance Study on the VM Startup Time in the Cloud, in: 2012 IEEE Fifth International Conference on Cloud Computing. Presented at the 2012 IEEE Fifth International Conference on Cloud Computing, pp. 423–430. https://doi.org/10.1109/CLOUD.2012.103

Nunes, J.P.K.S., Nejati, S., Sabetzadeh, M., Nakagawa, E.Y., 2024. Self-Adaptive, Requirements-Driven Autoscaling of Microservices, in: 2024 IEEE/ACM 19th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). Presented at the 2024 IEEE/ACM 19th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 168–174. https://doi.org/10.1145/3643915.3644094

(PDF) Enhancing Kubernetes Workload Efficiency Using Intelligent Scheduling Algorithms [WWW Document], 2025. . ResearchGate. URL https://www.researchgate.net/publication/390661357.

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. Association for Computing Machinery, New York, NY, USA, pp. 1–17. https://doi.org/10.1145/2741948.2741964