National College of Ireland

Machine Learning – Driven Auto Scaling in Kubernetes for Cloud Resource Optimization

MSc. Research Project
Cloud Computing

Ankur Shroff
Student ID: 23426659

School of Computing
National College of Ireland

Supervisor: Mr Abubakr Siddig

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ankur Shroff
Student ID:	23426659
Programme:	MSc. Cloud Computing **Year:** Jan 2025 - 1
Module:	MSc. Research Project Cloud Computing
Supervisor:	Mr. Abubakr Siddig
Submission Due Date:	11th December 2025
Project Title:	Machine Learning – Driven Auto Scaling in Kubernetes for Cloud Resource Optimization
Word Count:	6487 **Page Count:** 22

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: ……

Date: 11th Dec 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):	

Machine Learning – Driven Auto Scaling in Kubernetes for Cloud Resource Optimization

Ankur Shroff
23426659

Abstract

In Response to high fluctuation of user-traffic, cloud-computing services face the biggest challenge in managing dynamic resource allocation in Kubernetes as it uses default Horizontal Pod Auto-Scale (HPA) which only reacts to scale up or down depending on performance metrics based on limit threshold reached of traffic-load utilizing CPU, which leads to service degradation along with poor user experiences. In this research, auto scaling system design is more proactive and hybrid to overcome these limitations. This designed system that uses time-series forecasting model - Prophet, LSTM, ARIMA, which are trained on historical real-world cloud usage data – Alibaba 2018 Cluster and synthetic traffic-loads values, to predict future demand of system scaling. Thus, prediction is used to scale pods, where reactive component acts as a safety of scaling-down after traffic-load reduces. Performance of system was evaluated in Kubernetes environment based on 3 different phases from standard autoscaler to ML-predictions. To replicate the real-world traffic-load, tools are used to run the traffic-load on system to test the reliability of Kubernetes system based on LSTM-model with future demands of 10-second. Proactive scaler does improve resources allocation, resource utilization, anticipating traffic-load spikes and reducing the latency.

Keywords: Kubernetes(K8), Docker, Hey, Prometheus, Grafana, HEY, Push Gateway, Python, Django

1 Introduction

1.1 Background

The approach of cloud computing has revolutionized the development (Burns, et al., 2016) and deployment of the modern application in cloud infrastructure, the scalability and elasticity of the cloud infrastructure is very crucial (Al-Dhuraibi, et al., 2018), which allow systems to scale dynamically with the response of the user's demand. In Kubernetes the primary mechanism for scaling is default HPA, which only acts reactively (Kubernetes, 2025) after the traffic and CPU load impacts the system and functions by monitoring the system lagging load indicators – CPU utilization, which only starts scaling up after the metrics have reached or crossed the defined threshold. This delays the new pod replicas to initiate which then creates a gap for the application to process, leads in increasing more latency, service failure and bad user experience in real-world.

1.2 Importance

Standard HPA has reactive nature but has critical limitation in subject to highly volatile and rapid fluctuations in traffic-load patterns. At time, if high or sudden fluctuations of traffic-load

hits and the point where new scaling pods are fully active, create a significant amount of lagging. This lag combines of several delays including metrics collected over intervals, time taken to react by default HPA and pod implementation time – scheduling, pulling image and application startup.
In period of system lagging, the application is under critical supply which forcing existing active pods to handle the excess load, results in exceeding threshold capacity which leads to saturation of resources. While causes – higher latency, high request failure and bad user experience. Any modern application providing digital services from flash sales on e-commerce platforms to entertainment to critical news update, could cause user trust, engagement and revenue (Qu, et al., 2018). Thus, importance of smart and proactive scaling mechanisms which anticipate demand than just reactive to it.

1.3 Problem Statement and Objectives

- Problem Statement – For dynamic and volatile workloads, the standard reactive autoscaling inside Kubernetes environment is incapable of allocating resources (pods). This inherits the design limitation which leads to performance degradation and resource allocation over periods, creating gap between the theoretical promise of scaling and its practical implementation. (Nguyen, et al., 2019)
- Research Questions – How can time-series-based ML models improve auto scaling accuracy in Kubernetes to reduce cloud resource waste while maintaining SLA performance.
- Objectives – To provide solution to the research question, following objectives were establishes in this research:
    - Evaluate the performance and efficiency of baseline reactive Kubernetes HPA as a controller.
    - Establishing a theoretical performance by simulating a perfect forecast auto-scaler with 100% accuracy and future knowledge.
    - Implementing and evaluating a proactive auto-scaler driven by deep learning model – LSTM on Alibaba cluster dataset 2018.

1.4 Scope and Limitations

This conducted research is proof-of-concept inside a controlled experimental environment and the findings be interpreted using a local Kubernetes cluster (Docker Desktop). The workload is a specific on CPU based intensive web application. The traffic patterns been derived from real-world dataset Alibaba cluster 2018 dataset are reused in the environment.
The Study limits to time-series forecasting models and doesn't explore to other proactive techniques or ML models (schedule-based scaling or reinforcement-learning). The findings of this research provide a comparative analysis of the selected models under specific conditions and may not directly generalize all types of different applications with memory bound or input/output workloads or with different production environments without further configuration or validation.

1.5 Structure of Research

This research paper consists of – design, implementation and evaluation of this system, where objectives were:

- To design and implement a custom autoscaling system in Kubernetes which integrates predictive ML models.
- Train a time-series model on large real-world dataset (Alibaba Cluster 2018), ensuring it learns real-world patterns.
- Evaluate the system performance of different phases from reactive baseline to theoretical idea based on ML for proving the reduce resource wastage keeping the user experience intact by proactive hybrid scaling.

2 Literature Review

2.1 Paradigms of cloud Autoscaling

In cloud environment, the dynamic management of resources is controlled by various autoscaling strategies and are classified into three main paradigms – reactive, proactive and hybrid –

- Reactive scaling – Traditional and most widely implemented approach in the cloud environment in Kubernetes. The primary advantage of this approach is that it is simple and reliable – it scales based on real-time data and avoiding any assumptions. As the same suggests, it functions by reacting the real-time changes in the system utilization. Though, there is a limitation, documented by (Lorido-Botran, et al., 2014), is the latency between the workload changing and the reaction on scaling action. Which makes the system vulnerable while performing and results in performance degradation at sudden traffic-load increases.
- Proactive Scaling – This scaling method seeks to reduce high latency by forecasting future demand and get the resources ready in advance. This paradigm realizes the forecast through two methods, first is schedule - based scaling, which adjust based on the calendar or time during fixed hours or on a particular date. And the second is prediction – based scaling method, which involves mathematical models (Calheiros, et al., 2015) to forecast future-load, trained on historical patterns. Schedule – based are effective for high predictable (definite traffic-load to occur like Black Friday sale), loops in workload but lack in handling sudden spike in load, while prediction – based methods are smart, intelligent and have adaptive approach.
- Hybrid Scaling – Hybrid scaling is the combined of both previous paradigm – reactive and proactive autoscaling (Khan, et al., 2019). Proactive scaling forecasts to prepare for predicted traffic-load changes while retaining the reactive method to be used as a safety net to correct for prediction errors or respond to unforeseen spikes (due to various reasons like update, random offer, surprise product launch). In this research, architecture is developed for custom auto-scaler for hybrid system to be sure with more robust system.

2.2 Reactive Scaling in Kubernetes: The HPA

The default reactive scaling in cloud native ecosystem is Kubernetes HPA. The HPA controller does queries in certain period with inside Kubernetes metrics server for resources utilization data like CPU or memory usage. After which it applies a controlling loop algorithm, which is typically a ratio-based calculation –
(desiredreplicas = ceil[currentreplicas * (cuurentmetricvalue / desiredmetricvalue)],

To determine the needed replica count (Kubernetes, 2025). Even HPA is a powerful and essential tool for many applications, but it limits with sudden high traffic-load, which is the precise problem in this research.

2.3 Time-Series Forecasting for Workload Prediction

The victory of prediction-based proactive auto-scaler completely depends on precision and productivity of forecasting model. The job of anticipating future web traffic is an apt example of time – series forecasting problem. (Imdoukh, et al., 2020)

2.3.1 Statistical Methods: ARIMA

The Autoregressive Integrated Moving Average (ARIMA) model is a foundation of traditional time- series analysis (Box & Jenkins, 1970). It is an event of model's sequential structure by combining autoregression which depends on prior values, differencing to make series stable, and moving averages which could be dependent on previous forecast failures. ARIMA is well-recognized and can be efficient for models with clean, steady trends and repeating patters. However, it does have two limitations for this research. First it considers linear relationships in data which could make it inefficient for complex, non-linear data patters. Second it could require longer processing time, specifically for large datasets. ARIMA has been opted for this research as a standard baseline for conventional statistical model.

2.3.2 Decomposition Models: Prophet

Unlike standard statistical models, modern decomposition-based models have acquired a reputation due to the flexibility and performance they provide for real-world business time-series. An apt example is Prophet which is an open-source forecasting tool developed by Facebook's core data science team. As elaborated by the makers (Taylor & Letham, 2018), Prophet is an additive decomposition model that breaks down model into elements of trend, periodic fluctuations (example- daily or weekly), and holidays. This method makes model durable for any missing data or changes in trends and permit the model to handle nonlinear patterns. It's designed to precisely tackle routines found in web traffic data, making it suitable for research problems. Prophet was opted to signify sophisticated, domain centric forecasting model.

2.3.3 Deep Learning Models: LSTM

In past few years deep learning models specifically Long Short-Term Memory (LSTM) networks, have been implemented in time-series forecasting (Hochreiter & Schmidhuber, 1997). LSTM-models are a type of Recurrent Neural Network (RNN) which is built to handle long term dependencies and complex models and to have nonlinear types of relationship in sequential data. Although they have relative potential, they usually require huge amounts of training data (Toka, et al., 2020), heavy computing resources for training purposes, and advanced tuning of model parameters. As this is an experimental study, it requires focus on comparing easy to apply and easy to understand models, LSTMs resulting in far from scope.

2.4 The Alibaba-dataset and the Identified Research Gap

The considerable difficulty in investigating auto-scaling systems is the acquisition of workload data. The Alibaba Cluster Trace V2018 dataset (Alibaba Group, 2018), used for model training in this study, is a valuable resource. It consists of machine usage data from a production cluster

of over 4,000 machines covering 8-day period which provide precise look of real-world server behavior. Though in this research for Alibaba-dataset 2018, a research gap emerges from the data characteristics. Analysis of the Alibaba trace revealed the data is volatile over the long period (Guo, et al., 2019), and which has similar values and considered to be nearly similar traffic-load.

3 Research Methodology

3.1 Research Approach

Research applies statistical experimental research design (Moore, et al., 2013). A comparative analysis has been performed in 3 different phases, where each phase shows different adaptive scaling strategy. Phase 0/1 uses same workload, but phase2 is trained and performed with Alibaba real world dataset pattern. The method is designed in a way that it could completely reproduced whenever configurations and scripts are provided.

3.2 Experimental Environment

- Platform – A single-node Kubernetes cluster provisioned by Docker Desktop on a Windows machine. (Docker Inc., 2025)
- Application Workload – A session less application has been built with the Django framework. The application highlights a single HTTP endpoint (/) that executes high processing task to process computational works and generate CPU load. The application is created container-based using Docker.
- Load Generation – The load testing tool is used to produce simultaneous user traffic. It is executed inside a dedicated load generator pod placed in cluster to make steady response times.
- Monitoring Stack – The Kube-Prometheus-stack is deployed via Helm. This stack includes:
    - Prometheus: For time-series data collection and storage of all system and application metrics. (Prometheus Authors, 2025)
    - Grafana: For real-time visualization of set of results through dashboards. (Grafana Labs, 2025)
    - Pushgateway: To allow temporary scripts (like scaler and load-generators) to push metrics to Prometheus.
    - Prometheus Adapter: Used in Phase 0 to reveal custom Prometheus metrics to the Kubernetes HPA.

3.3 Dataset and Traffic Pattern

- Synthetic Data – Synthetic traffic-load has been created ranging from 100 to 1000 values to evaluate the performance of phase 0 and phase1for period of 60-minutes.
- Source Data – Research leverages the Alibaba 2018 dataset, an open-source dataset available, which consists of 8 days machine usage data from a big production cluster of size nearly 8gb file.
- Data Processing – Raw data is processed to separate CPU utilization (percent) for a single highly unstable machine. Data was recorded every 10-second and aggregated in new intervals to check maximum utilization every minute.

- Traffic Simulation – The CPU-utilization (from 0-95%) is converted to user load of maximum 2500-users.
- Test Traffic Pattern – Single extremely unstable nearly 70k rows detected from Alibaba 8-day dataset was visualized, which showcases volatile pattern in figure and used as the master traffic plan for all evaluation phases. This makes sure a fair and direct comparison.

3.4 Evaluation Metrics

To provide full-scale review, two primary categories of metrics are used:

- **Performance (User Experience):** Measured by client-side application processing delay, captured by hey load-generator. Average response time during maximum load is key metric of decline in performance, so lower values are better. (Emeakaroha, et al., 2012)
- **Resource Efficiency –** Measured by pod CPU-usages, this metric calculates the resource wastage for every-minute of experiment and average of low or excess CPU-usages of running pod, required to handle the current traffic load is high (above threshold limit), low(below 40%) or optimum(ranging 50-90%).

3.5 Experimental Phases

Research is structured in 3 different phases:

- Phase 0 Reactive HPA Baseline – Standard Kubernetes HPA configured to scale resources based on active traffic-load. This measures the performance of a standard, responsive system.

- Phase1 Perfect Foresight – Non-ML simulation where scaler has perfect, 100% accurate knowledge of traffic-load 2-minutes in advance. This theoretically showcases the best-case scenario.

- Training the 3 ML models and evaluating – 3 ML models were trained – Prophet, ARIMA, LSTM and comparing the model evaluation for best performance, selected with least-error while prediction.

  For performance evaluation of 3-ML model – Prophet, LSTM and ARIMA, following two standard metrices is calculated –
  - MAE (Mean Absolute Error) – It is an average magnitude of error in a set of prediction, which does not consider the direction. In simple understanding, it tells on an average, for how many requests was the model wrong by.
  - RMSE (Root Mean Square Error) – Scoring rule that measures the average magnitude of the error. Since the errors are squared before they are averaged, the RMSE gives a high relatively weight to large errors. This is very useful if the model has huge of misses or it failed to predict.

- Phase2 Proactive Scaling LSTM-model – Real-time inference system is implemented, trained on Alibaba-dataset to predict future-load. Unlike phase 1, phase 2 doesn't know future and dedicated brain pod monitors live traffic and feeds it into the LSTM-model in real-time and generates the predictions to scale pods in real-time dynamically with trained model using Alibaba-dataset.

4 Design

This research consists of several key component interacting within the Kubernetes clusters (Google Cloud, 2025) from standard reactive model to hybrid proactive model. (Django Software Foundation, 2025)
The architecture consists of four primary layers:

- Workload layer – A containerise CPU intensive simple Django web application that serves as the target.
- Simulation layer – A dedicated traffic simulator that plays like a real-world trace data (Alibaba cluster 2018) to mini user behaviour in real-time
- Monitoring layer – A Prometheus and Pushgateway setup that acts as a central system to fetch the metrics and display in Grafana dashboard aggregating real-time metrics.
- Control layer – Considered to be the brain, this layer performs the online inference and executes scaling of pods with Kubernetes API in LSTM ML model.

It can be understandable more clearly in the following conceptual diagram –

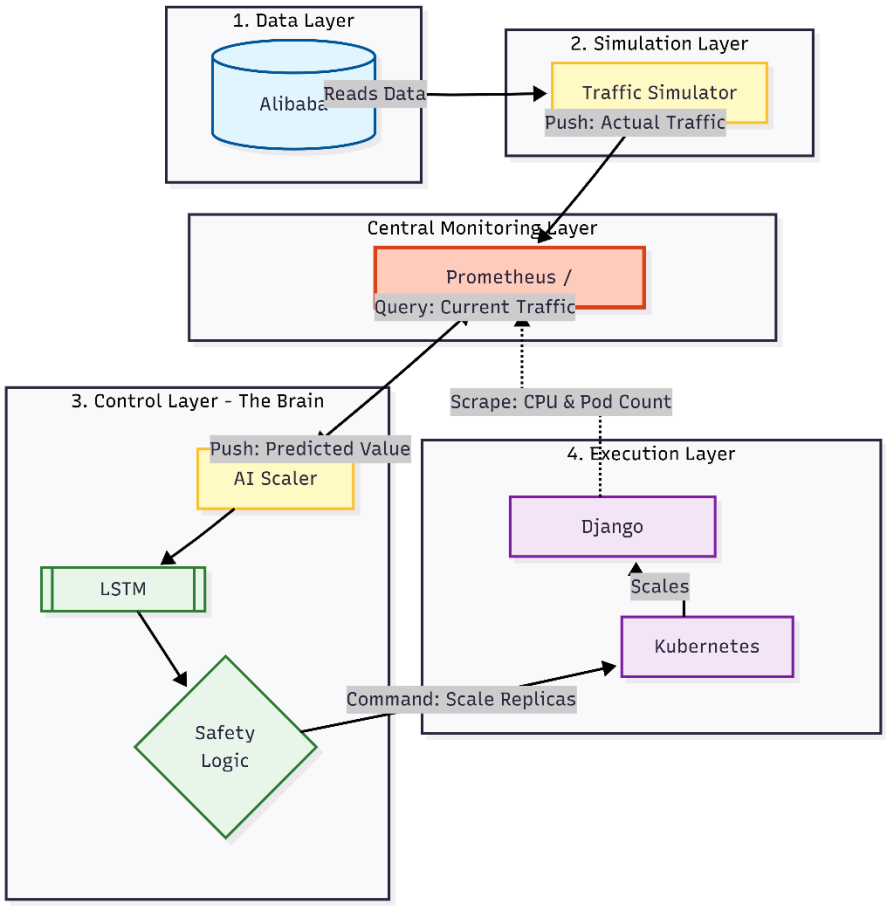

Figure 4.1 – Architectural diagram of ML model – LSTM

5 Implementation

This section explains implementation of various tools and strategies that were used to achieve desired results. In this research, core is pods scaling prior to need, which is configured on single

Kubernetes cluster in local machine, included Docker, Helm, Kubectl, Prometheus, Grafana, Pushgateway, Hey and more.

5.1 System Configuration, Docker, Kubernetes and tools integration

Following tools needed to be install in the machine before proceeding to the different phases:

- Python and Django – Python is used as the primary programming language for all scripting, including data processing, model training and proactive scaler logic. The Django framework was installed and was chosen to develop CPU intensive web application that was used for target workload. Python libraries included joblib, pandas, scikit-learn, pandas.
- Docker – Fundamental for containerization, the docker desktop for windows application was installed on the hosting machine to provide runtime engine for the container. Primary role to build the image of application from file – DOCKERFILE, a package of Django application and all related dependencies – Python, Gurnicorn web server into portable and self-contained unit which ensure the web-application runs similar in every phase.
- Kubernetes – Being an orchestrator for docker where all different phases of experiment were performed. The cluster was not created on cloud provider but was performed locally using Kubernetes engine inside docker desktop, this provide fully featured single -mode cluster. Kubectl command line was installed on local machine and served as primary interface for deploying the application, resource allocation and management, along with inspection of cluster.
- Helm – Helm CLI was installed locally as it's a package manager for Kubernetes, for managing the monitoring stack and its role was to automate the deployment of Kube-Prometheus-stack, which is a complex chart with huge inter-dependent Kubernetes resources.
- Prometheus and Grafana – A comprehensive monitoring system bases on the Kube-Prometheus-stack was deployed using the Helm. A critical implementation was developed, that to be used into two distinct configurations which enabled the Prometheus adaptor for all phases, which is sent to Grafana to display the performance, load and metrics in dashboard using all the values for monitoring the system and the load.
- Hey – Runs the traffic load pattern mimicking the real-world traffic pattern based on certain values provided using command line or with CSV file which mimics as real-world-load for applications.

5.2 Building Web Application

5.2.1 Application Logic

Core application was designed and build consuming CPU-intensive resource by performing factorial calculation of 1000. This computes the workload, ensuring, CPU-usages directly related with increase in traffic-load to process the application.

5.2.2 Containerization

Application was containerized using Docker. The dockerfile, a configuration for python based image including the dependencies from a requirements.txt file for Django web application,

copies the application entire code and the command to start webserver where further load-testing will be applicable and measure the performance.

5.2.3 Kubernetes Deployment

Containerized application is deployed to Kubernetes using deployment and service using the kubectl command inside the kind cluster named research-cluster to run the application. The file Django-app-development.yaml and Django-service.yaml files defines the required state which includes which container image to use and most important the resources allocations. These resource thresholds are essential for Kubernetes scheduler as well as for default HPA calculations of metrics.

5.3 Dataset Acquisition and Processing

The core foundation of any approach drive by ML is a data on which the model is trained, and for this research Alibaba 2018 cluster trace was selected as the source of the real-world traffic workload data record for 8 days for nearly 4000 machines but we are only processing for 1 machine for this research and performed in following steps –

- Downloading Data and handling – Alibaba is large dataset file in tar.gz format from a public repository, which was downloaded using python script download_dataset.py extracted csv file which is nearly 8-gb file size – machine_usage.csv, and performed the essential cleaning ensuring consistent and minimum error at the starting point.
- Data Transformation – CSV file machine_usage.csv have record of every 10-second interval data for thousands of machines, so processing entire dataset becomes challenge with-regards to memory. To overcome, processing done chucks. The scripts read the large data in small-chunks for only 1-machine usages – m_1. This technique of processing, successfully completed in local machine without limiting the system memory. Once done, a critical concern raises the CPU loads was utilized for how many pods from the single machine? So, the transformation was converted to traffic-load with maximum of 2500 for 100% CPU-usage.
- Visualizing – visualize the CPU pattern and traffic pattern (converted from CPU) shows violate fluctuations of resource usages in following figure 5.3 and 5.4

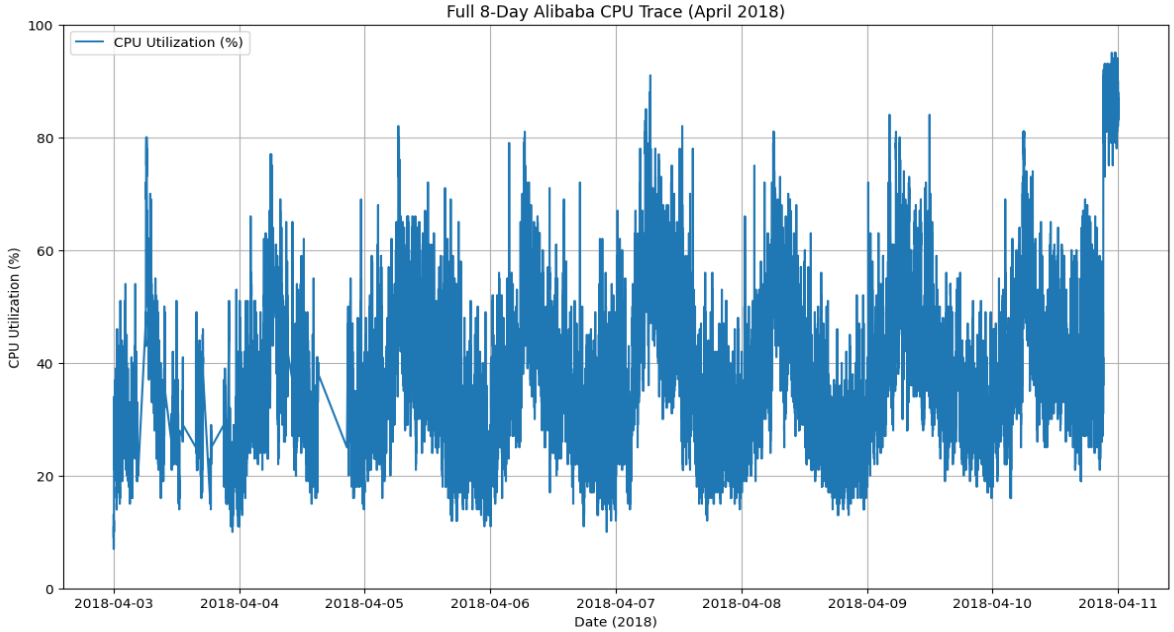

Figure 5.3, Visualization of Alibaba usages (CPU-usagess)

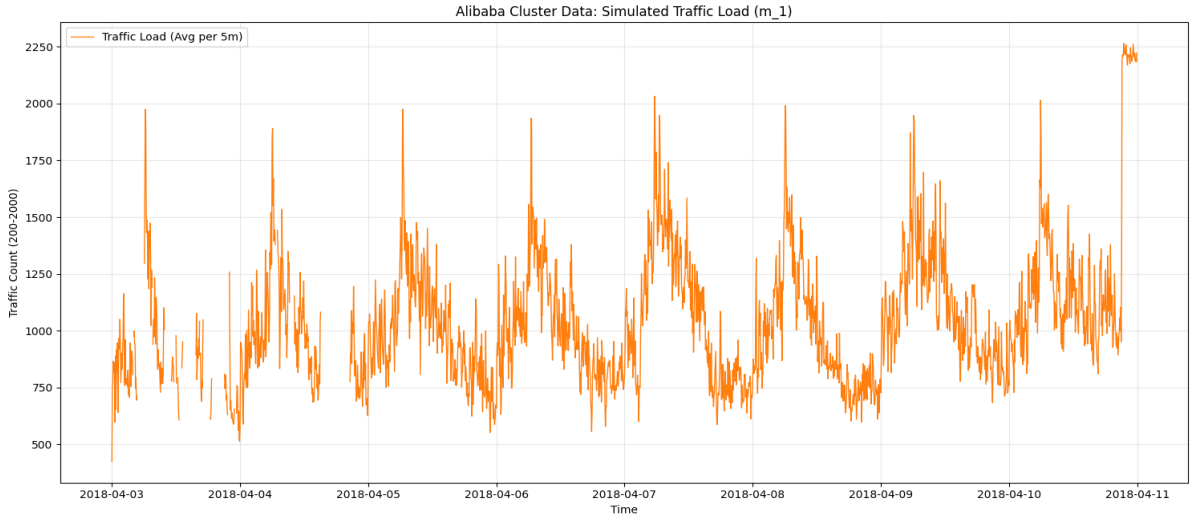

Figure 5.4, Visualization of Alibaba usages (CPU converted to Traffic-load)

5.4 Synthetic workload for initial evaluation

For initial analysis process of phase 0 and phase1, a synthetic data with maximum of 1000 users hitting the server for 60 minutes to test default HPA-phase 0 and perfect foresight-phase1. The python script prepare_phase0_synthetic_data.py, which created a csv while with random traffic-load processed by load generator pod and record the performance

5.5 Phase 0 – Baseline default HPA

Evaluation of default HPA of Kubernetes before evaluating predictive ML autoscaling model, it is important to test and establish a baseline performance, which serves as controller in this research, which represents standard, industry accepted approach for scaling inside Kubernetes environment. Main objective of phase 0, to deploy and measure the performance of default reactive HPA, configured to scale based on application traffic-load. By recording the performance of phase 0 under dynamic-load and benchmark against which the proactive systems in next phases are compared critically and fairly. Pods scales up when the performance reaches the define threshold and scales down after 60-seconds (configured manually, default is 5-minutes) depending upon the utilization on the CPU, which is predefined threshold, but pods are scaled in phase 0 based on the traffic-load defined with –

- Total (external metric value) = simulating the traffic from Pushgateway
- Target average = total average value, here 100 users per pod
- Actual pods = this is the ceiling function where the pods scale to the nearest value
- Calculation of actual pods = total / target average
- No limit to CPU-usages = Allowed to usage as many cores without limit but to scale based on traffic-load, not CPU-usage.

Consider – Traffic hits 588 (total), average is 100, so the actual pods will be 5.88, rounding to 6 actual pods.

5.6 Phase1 – Perfect Foresight

Phase1, establishes the ideal performance ceiling system with perfect knowledge of the future traffic demands, same csv used in phase 0 – predefined traffic-load for 60-minutes. Phase1 is not intended to be practical or real-world solution, but its main purpose is to create performance ceiling – best result that predictive ML system could be hoped to achieve. Performance created

a vital benchmark against the real-world ML-model for next phase that to be evaluated on efficiency and accuracy.
While implementing phase1, critical issue occurred, in initial stage the predicted pod scale-up before the actual traffic hits but when predicted to scale-down, pods were scaling-down before actual traffic-load is low. Which causes high traffic-load on active pods resulting in error or if talk about real-world scenario, its degrading the resources and bad user experience.
To resolve, hybrid system was developed where pods scale-up as predicted but scale-down after actual low traffic hits else, it will be maintaining the required pods based on 100 users per pod with no-limit to usage as many CPU-cores.

5.7 Training 3-ML models

After successfully benchmarking phase 0 and pahse1, 3-ML models – Prophet, LSTM, ARIMA were trained on process Alibaba-dataset (converted to traffic-load) for 80% of data and evaluated and compare best performing model with least error and use in phase2 for scaling pods. 3-separate file were created train_prophet.py, train_arima.py and train_lstm.py to train these 3 models and saving the trained file inside a folder model to further evaluate.

5.8 Phase 2 – ML base on Alibaba 2018 Cluster on LSTM-model

Core experimental phase of this research, transitioning from theoretical perfect-foresight model to real-time proactive autoscaling deployed in Kubernetes environment. LSTM-model was used as predictive engine because of high accuracy – low MAE during the offline evaluation, prior to live deployment and keeping sure that average resource utilization between 50-90% as if pods get a slight heavy load or enough pods are not active based on configures threshold the active pods can handle the load. Keeping 50 users per pod, limiting resource to half-a-core per pod.

5.8.1 Initial logic

Live implementation consists of following 2 independent, containerized process running parallel inside single Kubernetes pod load-generator –

- Traffic simulator – Python process responsible for real-world traffic-load which streams the test portion of Alibaba-dataset at 10-second interval, which performs 2 critical actions –
    - Metric Publication – Pushes current traffic value as metric (current_simulated_traffic) to Pushgateway which serves as real-world value for environment.
    - Load Generation – Executes hey load as subprocess, sending continuous HTTP request to target Django-app-deployment. Concurrent level of hey is set to current traffic value, ensuring application experiences real CPU pressure corresponding to publish metric.

- Predictive auto-scaler – Brain for autoscaling, having no direct access to future traffic-load and operates continuously in 10-second loop by sensing the metrics, memorizing the value and scales pods based on prediction from trained LSTM-model.

5.8.2 Predictive logic – Recursive forecasting

Single step prediction, 10-second ahead provides insufficient lead time for Kubernetes to create and assign new pods. To address this issue, muti-step recursive forecasting strategy was implemented –

- Recursive Forecasting – At each interval, the brain uses current 10-minute history to predict the traffic for T+10s, prediction is fed back into the historical data to predict T+20s, continuing for 6 steps to generate next 60-seconds trajectory.
- Peak Detection – System identifies maximum value in next 60-seconds future.
- Confidence buffer – Since LSTM-model isn't 100% accurate and is conservative in nature, so 10% confidence buffer is applied to predicted value, so for peak detection for future traffic, value is multiple by 1.10 to create buffered_prediction.

5.8.3 Hybrid safety logic for Scaling-up

Relying completely on predictive scaler carries high significant risk, if model incorrectly predicts a drop in traffic-load while actual-load remains high, system will scale-down prematurely, causing heavy load on active pods(resource-degradation), service-outage, bad user experience. To eliminate this risk, implemented a hybrid scale-up strategy that combines the ML-foresight and safety of reactive metrics, which is based on calculation –
Target_traffic = max(current_traffic, buffered_prediction)
Logic describes the system's behaviour in distinct scenarios –

- Proactive Scenario where prediction is greater than current – When ML predicts a spike, the buffered_prediction is higher. The system ignores the current low traffic and scales up immediately, ensuring pods are actively running before the load arrives.

- Safety Scenario where current is greater than prediction – If ML predicts a drop or predicts false negative, the current_traffic is higher. The system ignores the low prediction and maintains the capacity for active users.

5.8.4 Hybrid Safety net for scaling-down

Premature scaling-down or put heavy load on active pods, a 60-seconds cooldown buffer period was implemented that pods don't scale down proactively, it only scales-down after the actual traffic-load is low or reduced by maintaining a history of 60-seconds. The last value is pushed to pushgateway which ensures that even traffic drops, the resources are held for 60-seconds to confirm actual traffic has dropped and is not just a fluctuation, maintaining hybrid scaling-up logic if needed to scale-up or to maintain the existing.

6 Evaluation and Results

Evaluating and compare every phase, where phase was subjected to run for 60 minutes on the synthetic and real time predicted traffic to evaluate the performance of every phase.

6.1 Phase 0 – Baseline Performance

The performed metrics of the baseline Kubernetes HPA under a volatile synthetic workload ranging from 200 to 1000 concurrent users. The Grafana dashboard visualizes the 3 key metrics: Actual traffic (Green line), Total CPU-usages (orange line) and actual pod count (blue-line): –

Figure – 6.1 – Grafana Dashboard for phase 0 baseline performance.

- Analysis of scale-up latency (lag phase) – The primary limitation of reactive model is clearly visible demonstration during the traffic surges at timestamp 20:19 and 20:48, by green-line that traffic spikes rapidly from approx. 400 to 1000 users, however the blue-line, pod count exhibits a noticeable delay in reactive, taking several observation intervals to step-up to required capacity, reaching maximum for 10 pods consuming 20 cores (cluster consider threads as logic cores).

  During lag phase, existing pods are forced to absorb excess traffic, which results in resources degradation at earliest which cost hardware replacement and also throws error of server can't be reached which again causes bad user experience as show in figure 6.1

Figure 6.1 – Terminal Screenshot of Phase 0 where system lag

- Analysis of scale-down (Resource-wastage) –
  The graph highlights the inefficiency of HPA during the period of traffic decreasing. At timestamp 20:40 and 20:45, the traffic-load drops sudden, but still pod count – the blue-line remains elevated for a significant duration before steeping down, even though the configuration was made to only hold for 60-seconds.

6.2 Phase1 – Perfect Foresight

The figure 6.2 illustrates the performance of phase 1 system, which operates with perfect foresight looking 2 minutes ahead, combining with a hybrid scaling logic while scaling down. Graph plots the actual hybrid pod count (yellow-line), predicted pod count (blue-line), traffic-load (green-line) and the CPU-usages (orange-line) –

Figure – 6.2 – Grafana Dashboard for phase 1 Perfect foresight performance.

- Analysis of scale up – Define characteristics of this phase is observed during the major traffic surges, specifically at timestamp 22:23 and 22:50. In phase 0, the pod count lagged the traffic. In phase 1, the yellow-line – actual hybrid pods, scale-up in sync with or prior to green-line – traffic-load.
  Since, system already knew the traffic is coming 2 minutes in advance which is showcased by blue-line is towards left of traffic line, it is considered the required replicas of pod before the load arrives. Resulting, application handles the incoming spikes with fully active and running pods, eliminating the high latency and bad user experience.

- Analysis of hybrid safety – For premature scaling-down the hybrid approach was implemented where the pods scale down after actual traffic-load is reduced.
  - Observation – Between timestamp 23:00 and 23:05, the traffic green line remains high, though the predicted pods blue-line begins to drop frequently because it looks ahead and sees the upcoming downside.
  - System Response – If system would have followed only the prediction blue-line, then it would have scaled-down prematurely which might cause a crash. To overcome, actual hybrid pods yellow-line remain stable and high because of hybrid logic ignore the premature drop in prediction and maintains capacity based on current traffic-load as well.

6.3 Comparing the 3 ML model performance

6.3.1 Comparative table for 3 ML models

ML MODEL NAME	MAE (Low is better)	RMSE (Low is better)	APPROX – ERROR (%)	COMPUTATIONAL	VERDICT
ARIMA	406.84	489.34	20.3%	High (CPU intensive)	Poor, fails
PROPHET	299.87	463.79	15.0%	Medium	Good for trends, bad for sudden spikes
LSTM	70.45	98.66	3.5%	Low	Excellent, Production Ready

6.3.2 Analysis of the results

- ARIMA (The Statistical Baseline) –
    - Result: MAE: 406.84
    - Analysis: ARIMA (Auto Regressive Integrated Moving Average) performed the worst. With an MAE of ~407, the model essentially operates with a 20% margin of error. If the traffic is 1000, ARIMA might predict 600 or 1400.
    - Why it failed: Assumes data is linear and tries to project straight line or simple curve based on past data. Alibaba-dataset is highly volatile and non-linear; it has spikes caused by user behaviour. ARIMA cannot react fast enough to these changes, resulting in lagging prediction that always chasing the real value but never catching it.

- Prophet (Additive Regression Model) –
    - Result – MAE: 299.87
    - Analysis – Performed better than ARIMA but still had a 15% error rate.
    - Why it was mediocre: Designed by Facebook for business metrics (daily active users, sales) which have strong daily or weekly seasonality. It tries to smooth out noise to find general trend. However, in Kubernetes autoscaling, noise is signal. Need to react to spikes, not to smooth them out. Prophet smoothed outs spikes, causing it to under-predict during high-loads, which is dangerous for autoscaling.

- LSTM (The Recurrent Neural Network) –
    - Result: MAE: 70.45
    - Analysis: LSTM (Long Short-Term Memory), an overwhelming winner. An MAE of ~70 on a scale of 2000 represents an error rate of only 3.5%.
    - Why it won: LSTM is Deep Learning model specifically designed for sequence prediction. It has internal gates (Forget-Gate, Input-Gate) that allows –
        - Remember long-term trends (e.g., Traffic is generally higher in afternoon).
        - React to short-term sequences (e.g., Traffic jumped by 50 in the last 3 steps, it will likely jump again).
    - Research Implication: LSTM successfully learned the complex, non-linear usage patterns of the Alibaba cluster. The low RMSE (98.66) indicates that even when it was wrong, it wasn't by much—it rarely missed a massive hike.

6.4 Phase 2 – LSTM-model performance

Figure 6.4 is performance of phase2, which utilizes the LSTM ML-model for real-time online inference with hybrid safety approach for scaling-up and scaling-down. Figure provides view of interaction between the ML-prediction, scaling actions, and resulting resource utilization efficiency.

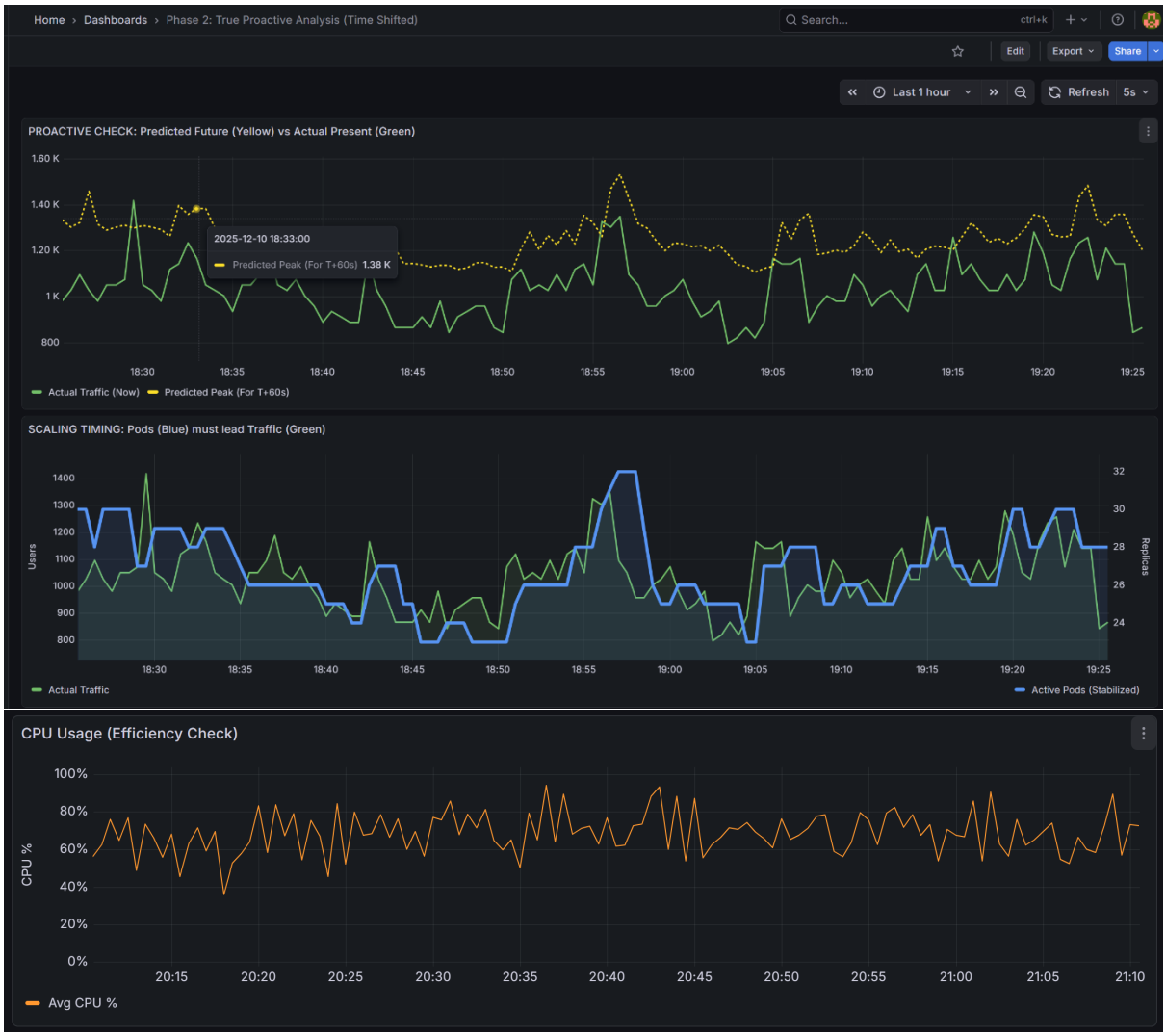

Figure 6.4 – Grafana Dashboard for phase 2 – LSTM ML model Performance

6.4.1 Analysis of predictive Accuracy

Proactive accuracy in graph compares the actual traffic (green-line) against the LSTM's prediction peak for next 60 seconds (yellow-line). Critical observation is that yellow-line consistently tracks slightly above green-line with following considerations –

- Safety Buffer – The model showcases a beneficial positive bias, for considering at timestamp 18:55, while actual traffic peaks at approx. 1350 users, model predicts 1500 users.
- Operational Implication – Behaviour validates LSTM-model has learned the volatile behaviour of the Alibaba-dataset by consistently predicting a higher value than the actual reality, model creates an automatic safety buffer. Though, actual traffic (green-line) ensures, pods are pre-scaled for current load but also for potential slight-hike, effectively eliminating risk of resource deficiencies.

6.4.2 Analysis of scaling timing (Lead vs Lagging)

Scaling timing in figure 6.4 confirms, resource optimized and ready for current and future-load with 10% safety buffer based on –

- Proactive Response – Scaling acts as ceiling that rises before the traffic (green-line) hits. At 18:57 as traffic hikes, pod count is already at 30 to stabilize.
- Hybrid Stability – Blue-line shows effectiveness of hybrid logic – (Target = max(pred, current). While traffic (green-line) goes back and forth frequently, still pod count doesn't thrash. It step-up aggressively to hold during low and prevents pod termination.

6.4.3 Analysis of resource efficiency (CPU-Health)

Figure 6.4 bottom graph shows average CPU-utilization of all active pods validating the logic of 50 users per pod as –

- Safe spot – CPU-usages (orange-line) fluctuates between 50-90%, to keep the resource utilization without getting over-used and handles sudden spikes.
- Efficiency – Rarely hits 100% (which causes higher latency) and below 40% (resource wastage). Confirms that proactive scaler, scales pods exactly when needed to keep load per contain within the optimal operating range.

6.5 Comparative analysis of 3 phases

Here will compare all the phases performed to determine the most viable autoscaling strategy for workload in cloud environments

6.5.1 Latency and responsive time

- Phase 0 – Failed to handle sudden hikes and scaling usually occurred after thresholds were breached, resulting in CPU-saturated, high-latency and bad user experience.
- Phase 1 – Proofed theoretical perfection with minimum latency but relied on knowledge of exact workload looked 2-minutes ahead.
- Phase 2 – LSTM successfully bridge the gap, performing online real-time every 10-second, minimum latency, and optimal CPU-usages. Figure-6.4 proofs scaling always occurred before and parallel to traffic fluctuations similarly mimicking the performance of phase1 but without future knowledge.

7 Conclusion, Discussion and Future Works

This section is consolidating findings from experimentation phase, discussing consequences and concluding the research by summarizing the key findings and suggesting areas for improvements in future.

7.1 Discussion of Findings

Results of experimentation of 3-phases shows clear picture of growth of autoscaling strategies within Kubernetes.

- Phase 0 – Baseline, using HPA clearly confirmed that reactive mechanisms are insufficient for unstable workloads. Grafana dashboards for phase 0 clearly outlined 'Lag Phase' where systems resource allocation marked lagging behind the actual traffic demand, leading to CPU saturation and potential service degradation. In contrast, pods were using high CPU-cores and were still lagging.

- Phase1 – Served as expected standard, setting up perfect foresight for performance. By providing scaler with advance knowledge of traffic, shows that minimum latency can be achieved as it removed the lack of resource which was seen-in phase 0. More importantly, displayed flaw in purely predictive systems that is early reduction of the resources. If model predicts traffic-drop too early, it can trigger a scale-down while the actual-load is still high, leading to immediate system overload. This led towards, hybrid Safety approach, which became essential part of final design.
- Phase 2 – Implementation, using the LSTM-model running in real-time online inference loop, confirmed efficiency of proposed solution. LSTM-model with its superior ability to capture complex, non-linear time-based dependencies achieved a low MAE of 70.45 (an accuracy of ~96.5%). This high-accuracy allowed brain-scaler to showcase stats which is very close to perfect foresight of phase1. Proactive scaling actions successfully assigned resources when forecasted traffic spikes, while integrated hybrid Safety approach (Target Pods = Max (Predicted Load, Current Load)) provided the necessary stability, preventing premature scale-downs and ensuring high availability. System effectively combined the best of both models, the flexibility of proactive-scaling with the reliability of reactive-safety checks.

7.2 Conclusion

Research has been successfully designed, implemented, and validated a ML-driven autoscaling system that addresses, inherent latency of Kubernetes default HPA. Research concludes that for dynamic and unstable workloads, a proactive, hybrid scaling strategy is clearly better than relying completely on purely reactive or ML-driven model.

Core contribution of research is successful deployment of real-time online engine in Kubernetes environment. LSTM-model proved to be the most effective model, precisely predicting real-world traffic patterns from Alibaba-dataset. Integration of Hybrid Safety approach was proven essential, creating a strong system that is both proactive and careful in assigning resources hence effective in both performance and stability.

Finally, research confirms the assumption that utilizing time-series forecasting can fill the gap between resource demand and supply in cloud-based environments, leading to more efficient resource-allocation, optimizing resource-utilization and creating an adaptable system with an enhanced user experience, especially during sudden spikes in traffic-load.

7.3 Future Work

While research serves the purpose, several potentials for future exists:

- Real-world E-commerce Applications – Instead of factorial 1000 logic, real-world e-commerce application and test load to check the performance in real-time with higher load.
- Real-Cloud Environment – Current simulations were run on local machine used as local cloud server; performance would be much better even with high traffic-load as the cloud server can use multiple clusters at same time.
- Reinforcement-Learning (RL): Future work could involve replacing the monitored LSTM-model with an RL agent, such an agent could learn the resource allocation by trial-and-error approach, where it can recognize for maintaining even further low latency and can be corrected for using too much resource allowing it to adjust to new traffic patterns automatically in real-time.

References

1. Al-Dhuraibi, Y., Paraiso, F., Djarallah, N. & Merle, P., 2018. Elasticity in cloud computing: State of the art and research challenges. *IEEE Transactions on Services Computing,* 11(2), pp. 430-447.

2. Alibaba Group, 2018. *Alibaba Cluster Trace Program, Available at:* https://github.com/alibaba/clusterdata

3. Box, G. E. P. & Jenkins, G. M., 1970. *Time Series Analysis: Forecasting and Control.* San Francisco: Holden-Day.

4. Burns, B. et al., 2016. Borg, Omega, and Kubernetes. *Communications of the ACM,* 59(5), pp. 50-57.

5. Calheiros, R. N., Masoumi, E., Ranjan, R. & Buyya, R., 2015. Workload prediction using ARIMA model and its impact on cloud applications' QoS. *IEEE Transactions on Cloud Computing,* 3(4), pp. 449-458.

6. Django Software Foundation, 2025. *Django Documentation, Available at:* https://docs.djangoproject.com/

7. Docker Inc., 2025. *Docker Documentation, Available at:* https://docs.docker.com/

8. Emeakaroha, V. C., Netto, M. A. & Calheiros, R. N., 2012. Towards autonomic detection of SLA violations in cloud infrastructures. *Future Generation Computer Systems,* 28(7), pp. 1017-1029.

9. Google Cloud, 2025. *Best practices for running cost-optimized Kubernetes applications, Available at:* https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture

10. Grafana Labs, 2025. *Grafana Documentation, Available at:* https://grafana.com/docs/

11. Guo, J., Chang, Z., Wang, S. & Ding, H., 2019. *Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces.* s.l., IEEE, pp. 1-10.

12. Hochreiter, S. & Schmidhuber, J., 1997. Long short-term memory. *Neural Computation,* 9(8), pp. 1735-1780.

13. Imdoukh, M., Ahmad, I. & Alqerem, A., 2020. Machine learning-based auto-scaling for containerized applications. *Neural Computing and Applications,* Volume 32, pp. 9745-9760.

14. Khan, A. A., Zakarya, M. & Khan, R., 2019. A novel hybrid auto-scaling method for cloud computing. *Cluster Computing,* 22(3), pp. 845-857.

15. Kubernetes, 2025. *Horizontal Pod Autoscaler,* https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

16. Lorido-Botran, T., Miguel-Alonso, J. & Lozano, J. A., 2014. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing,* 12(4), pp. 559-581.

17. Moore, L., Bean, K. & Ellahi, T., 2013. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Cloud Computing,* 2(1), pp. 1-15.

18. Nguyen, T. T., Yeo, C. S. & Buyya, R., 2019. Elastic resource scaling in the cloud: A review of techniques and research challenges. *IEEE Communications Surveys & Tutorials,* 21(4), pp. 3179-3205.

19. Prometheus Authors, 2025. *Prometheus: From metrics to insight, Available at: https://prometheus.io/docs/*

20. Qu, C., Calheiros, R. N. & Buyya, R., 2018. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys (CSUR),* 51(4), pp. 1-33.

21. Taylor, S. J. & Letham, B., 2018. Forecasting at scale. *The American Statistician,* 72(1), pp. 37-45.

22. Toka, L., Dobreff, G., Fodor, B. & Sonkoly, B., 2020. *Machine learning-based scaling of containerized applications.* s.l., IEEE, pp. 1-6.