Autonomous Upscaling Reinforcement Agents - AURA

CSD415 Project Phase I

Project Report submitted in partial fulfillment for the degree
B.Tech in Computer Science and Engineering

Submitted by

Eric Thomas K (Reg No: MUT22CS056)
Vaishnav S Nair (Reg No: MUT22CS146)
Navaneet Krishnan R Varma (Reg No: MUT22CS104)
Pranav R Mallia (Reg No: MUT22CS116)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(Affiliated to APJ Abdul Kalam Technological University)

October 2025

Muthoot
Institute of Technology & Science
Autonomous
RELENTLESS PASSION FOR EXCELLENCE

Kochi, Kerala - 682308
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project report entitled **"Autonomous Upscaling Reinforcement Agents (AURA) "** submitted by **Eric Thomas Kochupurakal (MUT22CS056),Vaishnav S Nair (MUT22CS146), Navaneet Krishnan R Varma (MUT22CS104), Pranav R Mallia (MUT22CS116)** of Semester VII is a bonafide account of the work done by them under our supervision during the academic year 2025-26.

Guide(s)	**Coordinator(s)**	**Head of the Department**
Dr. Sheena Kurian K	Jisha James / Sneha S	Dr. Anishin Raj M M
Asst. Professor	Asst. Professor	Professor
Dept. of CSE	Dept. of CSE	Dept. of CSE

Acknowledgement

This project is the result of my hard work wherein we have been helped and supported by several persons and institutions directly and indirectly. Now it is the time to acknowledge their contributions.

Foremost, we would like to express our sincere gratitude to our project guide, **Asst. Prof. Dr. Sheena Kurian K** for her continuous encouragement, invaluable guidance, motivation and enthusiasm throughout our project. We appreciate her sincere help in terms of patience, time and ideas so as to make us stimulating and productive.

Our sincere gratitude to **Dr.Anishin Raj M M**, Head of Department during our course period 2022-2026. We would also like to thank the project coordinator **Asst. Prof. Jisha James/Asst. Prof. Sneha Sreedevi** for critically assessing the work and giving valuable suggestions. We would like to acknowledge management of Muthoot Institute of Technology & Science for providing academic support to complete our project.

In our daily work we have been blessed with a friendly and cheerful group of fellow students. I am/We are very much thankful to all the members of S7 CSE.

We would like to express our sincere gratitude to all teaching and non teaching staff of Computer Science and Engineering Department for providing good environment.

Besides this, several people have knowingly and unknowingly helped in the successful completion of this project. We express our sincere gratitude to all of them.

Eric Thomas Kochupurakal
Vaishnav S Nair
Navaneet Krishnan R Varma
Pranav R Mallia

Abstract

When apps are constructed using microservices, managing system stability becomes complex. Traditional auto-scaling systems, such as the Kubernetes Horizontal Pod Autoscaler, are not responsive when usage spikes; they use simple metrics such as CPU or RAM. Consequently, resources are not always utilized efficiently, performance fluctuates, moreover responses lag during high-traffic load events. Modern technologies often tries fixing problems in isolation, yet fail to model inter-service dependencies. Because the services are not working with each other, scaling decisions are uncoordinated, delays accumulate, then expenses increase. Project AURA proposes an alternative approach – a multi-agent system collaborate as a collective, fine-tuning Kubernetes through shared effort instead of solo actions. Services evolve together, yet act independently once deployed - they learn from each other, then operate independently. During its training phase, AURA faced realistic hurdles like slow startup, scrape lag, also variable workloads within a typical setup. Thus, the system operates on rewards, carefully weighing cost savings (α) against swiftness and responsiveness (β), all while ensuring steady, reliable outcomes (γ). After learning, the programs run inside lightweight containers on a personal Kubernetes system. They monitor system state with Prometheus metrics, then pre-allocate resources to avoid SLA violations. In essence, AURA shows that letting multiple automatic tools collaborate outperforms traditional methods – meaning lower costs, steadier operation, and show an improvement over HPA.

Contents

List of Figures

List of Tables

List of Abbrevations

AURA	Autonomous Upscaling Reinforcement Agents
HPA	Horizontal Pod Autoscaler
VPA	Vertical Pod Autoscaler
SLA	Service Level Agreement
CPU	Central Processing Unit
RAM	Random Access Memory
AI	Artificial Intelligence
MARL	Multi-Agent Reinforcement Learning
RL	Reinforcement Learning
HRL	Hierarchical Reinforcement Learning
DRL	Deep Reinforcement Learning
LSTM	Long Short-Term Memory
GNN	Graph Neural Network
FIRM	Deep RL-based VM Autoscaling Framework (Mao et al., 2016)
AWARE	DRL with Workload Prediction Framework (Wang et al., 2018)
FIFER	Hierarchical RL Framework for Serverless Computing (Malla et al., 2020)
SPE-HPA	Self-Predictive Enhanced Horizontal Pod Autoscaler (Giménez et al., 2022)
DeepScale	Transfer Learning + DRL Framework (Yan et al., 2020)
H-Scale	Hybrid Reinforcement Learning Autoscaler (Zhang et al., 2021)
KubeKnots	GNN + DRL-based Topology-Aware Autoscaling Framework (Boden et al., 2023)
α	Cost Savings Weight in Reward Function
β	Swiftness and Responsiveness Weight
γ	Stability and Reliability Weight

Chapter 1

Introduction

This project introduces a cooperative, multi-agent reinforcement learning system designed to overcome the high costs and poor performance associated with uncoordinated microservice autoscaling.

1.1 Motivation

The move from monolithic applications to microservice architectures has become the standard for building scalable software. It provides clear benefits: independent deployment, technological flexibility, and fault isolation. This architecture creates issues for those in charge. A service's speed isn't just about itself anymore - it relies heavily on how well everything connected to it functions. Traditional auto-scaling systems - consider the standard Kubernetes HPA - will not be stable in a complex interconnected system where everything influences everything else. Their limitations provide the primary motivation for this research:

- Reactive and Myopic: Standard autoscalers are reactive in nature. They only scale after a simple metric , such as CPU utilization, has already crossed a predefined threshold. This will lead to periods of high latency and service degradation for the end-user. Furthermore, they are myopic, focusing on a single service without any awareness of the system-wide consequences.
- Uncoordinated Scaling: The core inefficiency stems from a lack of coordination. An HPA might correctly scale a frontend service to meet a traffic spike, but in doing so, it overloads the un-scaled database or backend service. This "fix" simply moves the bottleneck, creating a new failure point and resulting in latency.
- Financial and Performance Costs: This reactive approach results in two critical issues:
    1. Wasted Spend: Cloud money often vanishes because systems need a reserve of power - a safety net - that frequently goes unused. This preemptive allocation results in hefty bills for what isn't actually consumed.
    2. Poor User Experience: When lots of people use it at once, the system gets overloaded, then stalls, trying to keep pace.

    Even existing AI-based autoscaling solutions (as seen in our literature review) treat the system as a single entity, failing to model it as a team of cooperative agents. This creates a clear and

urgent need for a smarter, cooperative approach that can learn these complex inter-service dependencies and coordinate scaling decisions proactively.

1.2 Problem Statement

Traditional autoscaling methods, which treat microservices as isolated silos, fundamentally fail to manage the complex inter-service dependencies inherent in modern cloud applications, leading to systemic performance failures and inefficient resource allocation which is directly addressed by AURA by developing a cooperative, multi-agent reinforcement learning system designed to learn and execute coordinated scaling policies.

Chapter 2

Literature Review

Should Include all the papers used while doing presentation

Table 2.1: A Review of Existing Autoscaling Approaches

Existing Work	**Methodology**	**Advantages / Contribution**	**Framework / System**
Mao *et al.*, 2016	Deep Reinforcement Learning (DRL)	Pioneer work demonstrating that DRL can outperform traditional threshold-based rules for virtual machine allocation and autoscaling.	**FIRM**
Wang *et al.*, 2018	DRL with Workload Prediction	Combines an LSTM-based workload predictor with DRL to enable proactive, latency-aware scaling decisions.	**AWARE**
Malla *et al.*, 2020	Hierarchical Reinforcement Learning (HRL)	Introduces a two-level RL structure to manage both function placement and instance counts for serverless computing environments.	**FIFER**
Giménez *et al.*, 2022	Q-Learning + Prediction	Enhances the Kubernetes HPA using Q-Learning and workload forecasting to improve resource utilization and response time for web applications.	**SPE-HPA**
Yan *et al.*, 2020	Transfer Learning + DRL	Uses transfer learning to pre-train a DRL model on multiple workload types, enabling faster adaptation and improved generalization for new applications.	**DeepScale**
Zhang *et al.*, 2021	Hybrid RL (Model-based + Model-free)	Combines model-based simulation learning with direct reinforcement learning to enhance sample efficiency and scalability in cloud environments.	**H-Scale**
Boden *et al.*, 2023	Graph Neural Networks + DRL	Represents microservice dependency graphs using GNNs to perform topology-aware scaling decisions, improving coordination between services.	**Kube-Knots**

2.1 Paper 1, should include title, author name, published year

2.1.1 Summary

summary of paper 1

2.1.2 Methodology

2.2 Paper 2

2.2.1 Summary

summary of paper 2

2.2.2 Methodology

2.3 "Optimizing Resource Management in Kubernetes: A Study of Auto-Scaling and Load Balancing Approaches", A. Gogineni, International Journal, 2024.

2.3.1 Summary

The research explored Kubernetes resource management, focusing on its automatic scaling tools - HPA alongside VPA. It discovered that these scalers often clash when dealing with intricate microservice apps; therefore, some parts of an application might falter despite overall good performance from others. The paper's primary contribution is a comparative experiment that benchmarks traditional autoscalers (HPA, VPA) against an intelligent model. The goal is to demonstrate that a coordinated approach can simultaneousl reduce Service Level Agreement (SLA) violations and minimize cloud costs.

2.3.2 Methodology

The paper's methodology is experimental. The authors construct a representative three-tier microservice application (frontend, backend, and database) deployed on a Kubernetes cluster. This application is then subjected to a dynamic, realistic workload designed to trigger the "HPA miscoordination" problem—a scenario where the frontend scales out, overwhelming the unscaled backend. They measure the performance of HPA, VPA, and their proposed intelligent model. The comparison is based on three key metrics: total SLA violations (measuring performance), CPU efficiency (measuring utilization), and cost efficiency (measuring waste).

2.4 "A Survey of Autoscaling in Kubernetes",M.-N. Tran, D.-D. Vu, and Y. Kim, 2022

2.4.1 Summary

The work explores what's been studied regarding automatic scaling. It groups together Kubernetes' native auto-scalers - Horizontal Pod Autoscaler, Vertical Pod Autoscaler and Cluster Autoscaler - then gathers metrics from many experiments. These reveal a key drawback: they mostly just respond to problems once they happen, based on set limits. The study traces how automatic scaling has developed, sorting methods into those that forecast demand using past patterns, alongside ones employing artificial intelligence to learn and adjust. Importantly, this research backs up why we're here today - it shows where our MARL method fits among newer strategies.

2.4.2 Methodology

As a survey paper, its methodology is a systematic literature review. This research explores Kubernetes autoscaling through a thorough look at existing studies - both scholarly work alongside insights from the field. It collects then examines numerous publications. Papers get sorted by what they fundamentally do – how they react, forecast, or utilize machine learning – alongside which measurements matter (like speed, processor use, or specialized tests) then by the issue they tackle - whether improving function, lowering expenses, or conserving power. Reviewing these studies reveals a couple persistent hurdles: the paper identifies the lack of coordination and the "sim-to-real" gap as two of the most significant unsolved problems in the field.

2.5 "A Hierarchical Reinforcement Learning-based Autoscaler for Microservices in Kubernetes", X. Zhao, C. Wu, Z. Zheng, Z. Li, and J. Liu, Dec. 2022.

2.5.1 Summary

This paper proposes a novel approach to coordination via RL. It suggests typical MARL systems falter when dealing with numerous tasks; therefore, they present a layered, Hierarchical Reinforcement Learning (HRL) structure for improved performance. A main controller , "high-level" agent (or "Central Critic") grasps the big picture - overall speed, expenses - then gives tasks to smaller worker programs. Those workers handle adjustments to separate parts of the system, getting credit not for what they do directly, yet for meeting the targets given by the controller.

2.5.2 Methodology

The authors design and implement a custom HRL framework. The problem is decomposed: the high-level agent observes the global state (e.g., overall request rate) and its action is to distribute a resource budget or set latency targets for the low-level agents. The low-level agents observe only their local state (e.g., their pod's CPU, queue length) and their action is to scale up or down. This hierarchical policy is trained in a simulated Kubernetes environment. The reward function is structured to teach the high-level agent to make wise strategic decisions and the low-level agents to execute those decisions efficiently, proving that this separation of concerns leads to a more stable and scalable policy than a non-hierarchical RL approach.

Chapter 3

Proposed Project

3.1 Objectives

Clearly defined 3-4 statements.Expected to define the product what your system is going to produce in the end

3.2 Proposed solution

Chapter 4

Project Design

4.1 System Architecture

not only the architecture diagram, but include a description of the same.

Figure 4.1: Muthoot Logo

4.2 Modules

Also describe all the algorithms used in each module

Table 4.1: Overview of DC microgrid systems

DC grid connection	Voltage level	ESS connection	Stability	Reliability
Single-bus Unregulated DC grid	12, 24, 48	Direct	High	High
Single-bus regulated DC grid	12, 24, 48, 380	using converters	Medium	Medium
Multi-bus regulated DC grid	48, 380	using converters	Medium	Medium
Ring based DC grid	24, 48	using converters	Medium	High
Zonal based DC grid	380 and higher	Direct or using converters	Medium	High
Mesh based DC grid (MTDC)	380 and higher	Direct or using converters	Medium	High

4.3 Data Flow Diagrams

Level 0 and level 1 is mandatory. other levels may be included as per the project. here also description is mandatory for describing the DFD.

4.4 System Requirements

4.4.1 Hardware

4.4.2 Software

Chapter 5

Project Schedule

Gantt chart Required both 1st phase and 2nd phase

Chapter 6

Conclusion

in single paragraph

Chapter 7

Bibliography

`Include 10-15 reference papers`

1. A. Gogineni, "Optimizing Resource Management in Kubernetes: A Study of Auto-Scaling and Load Balancing Approaches," *International Journal*, vol. 7, no. 11, pp. 241, 2024. ISSN: 2348-9510.

2. M.-N. Tran, D.-D. Vu, and Y. Kim, "A Survey of Autoscaling in Kubernetes," *Proc. of the 2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2022.

3. X. Zhao, C. Wu, Z. Zheng, Z. Li, and J. Liu, "A Hierarchical Reinforcement Learning-based Autoscaler for Microservices in Kubernetes," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4668–4683, Dec. 2022.

4. Author, "Title of the work", International journal name, Vol no, page no,year

5. Author1, Author2 et al; "Title of the work", International journal name, Vol no, page no,year

6. Author, "Title of the work", In proc. of international conference in place, page no,year