National College of Ireland

Dynamic Resource Scaling for Microservices: A Machine Learning-driven Predictive Approach

MSc Research Project
MSc in Cloud Computing

Akshay Joshi
Student ID: 22137696

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akshay Joshi
Student ID:	22137696
Programme:	MSc in Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	30th January 2024
Project Title:	Dynamic Resource Scaling for Microservices: A Machine Learning-driven Predictive Approach
Word Count:	XXX
Page Count:	18

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:	Akshay Joshi
Date:	30th January 2024

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):	

Dynamic Resource Scaling for Microservices: A Machine Learning-driven Predictive Approach

Akshay Joshi
22137696

Abstract

In my research on "Dynamic Resource Scaling for Microservices: A Machine Learning-driven Predictive Approach," the focus is on optimizing microservice-based applications by predicting and adjusting resource needs using cloud auto-scaling mechanisms. The machine learning models, including Multinomial Logistic Regression and Linear Regression, are trained on historical workload data, specifically the Netflix NDbench dataset.

Following the training, microservices are deployed on AWS Lambda with AWS API Gateway and a Load Balancer for efficient management. The machine learning outcomes categorize CPU usage into High, Normal, and Low, stored for subsequent actions. To operationalize these predictions, a dedicated webpage is developed, displaying machine learning results and Lambda function details through AWS SDK.

The critical component of the research involves autoscaling by dynamically updating the concurrency of the Lambda function. When the machine learning model predicts high CPU utilization, the concurrency is increased, allowing the Lambda function to handle a higher number of simultaneous requests. Conversely, during low CPU usage, the concurrency is decreased to optimize resource allocation.

For an in-depth exploration of the autoscaling methodology and its integration with machine learning predictions, further details are provided in subsequent sections.

1 Introduction

In the software development, microservices play a crucial role. Rather than constructing a single large application, developers create small, specialized components known as microservices, each dedicated to specific tasks such as user authentication, email sending, or payment management. These microservices collaborate like a team, collectively contributing to the functionality of the application. Visualize a group of efficient, specialized workers (microservices) in a company, each handling distinct tasks swiftly. Their workload is dynamic, making it challenging to predict when they'll be busy or idle.

To address this challenge, it becomes imperative to allocate the right resources to these microservices at the right time. Traditional approaches involved servers reacting to past workloads, akin to driving a car while only seeing what's behind and not anticipating what lies ahead. Microservices, designed for easy adjustment and independent growth, offer scalability and efficiency. Interviews in the software/IT field revealed that 90% of

respondents emphasized easy scaling and independent extension, while only 15% cited the ability to reuse microservices as a primary motivator.Zhou et al. (2023)

Microservices, although providing flexibility, create variable workloads on servers, especially during sudden spikes in usage. The conventional method of adjusting server resources proves inefficient. This research seeks to address this inefficiency using machine learning. The goal is to create a system capable of predicting when microservices will be busy or idle, enabling proactive resource allocation and avoiding unnecessary expenses. The proposed solution, termed "Dynamic Resource Scaling for Microservices" (DRSM), acts as an intelligent traffic manager for servers, ensuring optimal resource allocation for incoming tasks.Hossain et al. (2023)

The key challenge is predicting future workloads and preparing servers in advance without incurring unnecessary expenses or causing delays when additional resources are required. The research aims to establish a predictive framework for efficient resource scaling, enhancing company efficiency and cost savings in the long run.

My goal is to use machine learning to identify a better approach to handle this. I have developed a system that can anticipate busy periods and provide intelligent workers with the necessary resources in advance. By doing this, I can make sure they always have what they need to perform at their best work and prevent wasting money on extra tools. In the long run, I think this will improve a company's operations and result in cost savings. But how can I accomplish it without spending money on unnecessary instruments or holding up their needs for additional assistance?

I have a creative method for predicting future workloads and preparing equipment ahead of time in order to overcome this difficulty. The solution to this problem is "Dynamic Resource Scaling for Microservices" (DRSM). It's similar to having an extremely intelligent manager overseeing the servers, ensuring they have the appropriate equipment to complete every work. The main goals of this research are to increase programme efficiency, reduce costs, and make it work smarter.

1.1 Research question

Research Question:- How can a microservice-based architecture with machine learning algorithms be utilized to predict and adjust resource demands of microservices in a cloud environment, enabling proactive and efficient autoscaling in response to dynamic traffic on backend systems?

1.2 Research objectives

The main goal of this project is to create and put into practice a microservice-based architecture that incorporates machine learning methods in order to forecast and modify microservice resource demands in a cloud environment. Enabling a proactive and effective autoscaling method that adapts dynamically to variations in backend system traffic is the main goal. Particular objectives consist of:

1. **Architectural Design:** Designing a microservice-based architecture that accommodates the integration of machine learning algorithms for resource prediction and autoscaling.

2. **Machine Learning Model Development:** Developing robust machine learning algorithms capable of accurately predicting resource demands based on dynamic

traffic patterns and workload variations.

3. **Autoscaling Implementation:** Implementing a proactive autoscaling mechanism that leverages the machine learning predictions to dynamically adjust the allocation of resources to microservices, both vertically (hardware resources) and horizontally (microservice replicas).

1.3 Document Structure

This research report is divided into several sections. in 2 i have reviewed the related paper, In secction 3 I have explain about Normalization and Classification of System Metrics , Data Preparation , Architecture for Load Management and Prediction in Serverless Environments , In section 4 I have talked about which machine learning model I have used. In section 5 I have explain how I have did the end-to-end process of my research. In section 6 I hav discuss about the case study Finally in section 7 I have successfully answer my research question.

2 Related Work

Xu et al. (2022) examines the difficulties with load balancing and auto-scaling in cloud-based container microservices in detail. The significance of these methods in enhancing user performance and Quality of Service (QoS) is covered. The article discusses container microservices, cloud-based microservices, and microservice architecture. It highlights how load balancing and auto-scaling strategies must be integrated to get the best performance and resource use out of microservices.

Xu et al. (2022) focuses on analysing previous research and giving an in-depth explanation of the difficulties with managing load and auto-scaling in cloud-based container microservices. There are no new ideas or contributions to load balancing and auto-scaling in this work. Rather, it gathers and evaluates already published research in the area. Realistic ApplicationXu et al. (2022)provides future work on load balancing and auto-scaling hybrid capabilities, but it lacks of case studies or specific examples of real-world implementations. Comparative Analysis: Because the research does not compare various load balancing and auto-scaling strategies, evaluating their efficacy in particular situations is difficult. nevertheless, in contrast Xu (2022) presents CoScal, a cutting-edge multifaceted scaling technique for microservices in cloud computing environments. To handle resource scaling issues, it incorporates brownout approaches, vertical scaling, and horizontal scaling. The advantages of each technique, such as fine-grained control, self-adaptability, and high availability, are leveraged in this study. CoScal employs precise workload prediction for effective resource management and a scaling technique based on reinforcement learning for adaptive decision-making. The usefulness of the suggested method for enhancing resource scalability in cloud data centres is assessed using realistic traces and a prototype system. Xu (2022) focuses on introducing CoScal, a novel multifaceted scaling solution that employs reinforcement learning-based decision-making to tackle the resource scaling issue in microservices. In order to provide adaptability in dynamic contexts, the study provides a novel method of merging multiple scaling strategies and utilising reinforcement learning for optimised decision-making. To ensure practical relevance, the paper offers assessments using realistic traces from Alibaba and a prototype system. As though Xu

(2022) draws attention to the differences from earlier work; yet, a more thorough comparison with current scaling methodologies and algorithms would be beneficial.

Comparison and Conclusion: While they both discuss scaling issues with cloud-based microservices, the papers' contributions and areas of attention differ. Xu et al. (2022) gives a thorough analysis of the research that has already been done on load balancing and auto-scaling without proposing any new methods. On the other hand,Xu (2022) presents CoScal, a revolutionary multifaceted scaling technique that makes use of reinforcement learning to make decisions and accurately estimate workloads. Xu (2022)is more beneficial because of its creative methodology and useful analysis. Both papers advance our knowledge of microservices scaling strategies and offer suggestions for future study and development directions.

Papers offer original strategies to deal with particular problems in their fields. Chung (2023) concentrates on task-based parallel applications' load balancing, while Xu (2022) addresses the scaling of resources in microservices.

Both publications use assessments and experiments to demonstrate the advantages of their suggested methodologies. To show how effective their approaches are, they provide the outcomes of their experiments. Both articles emphasise how critical it is to base evaluations on actual traces or benchmarks and take into account real-world events.

Domain and Problem Scope:

Chung (2023): It focuses on load balancing in distributed memory machines task-based parallel applications. Xu (2022): In cloud computing environments, it deals with resource scaling in microservices.

Approach and Techniques: Chung (2023): The authors suggest a proactive method of load balancing that makes use of prediction models and task characterization to inform balancing strategies as they execute. Xu (2022) In order to optimise resource scaling for microservices, the authors present a comprehensive scaling approach called CoScal, which includes brownout approaches, vertical scaling, and horizontal scaling.

Methods and Algorithms: Chung (2023): The proactive load balancing approach in Xu (2022) is based on characterizing task features, learning load values, and predicting execution times. Xu (2022): In CoScal makes use of scaling methods based on reinforcement learning to maximise the performance of microservices according to different criteria.

Use Cases: Chung (2023): Micro-benchmark experiments and a practical application of adaptive mesh refinement simulation are included in the evaluation Xu (2022) The evaluation is performed using a prototype system and realistic traces derived from Alibaba.

Conclusion:-Both papers address load balancing and resource scaling issues and offer innovative solutions to their respective fields. As though Chung (2023) suggests a proactive method for parallel task applications, Xu (2022) presents a complex method of microservices scaling in cloud environments. Both papers use experiments to show the effectiveness of their methods, but they each have certain aspects that might be improved upon, such going into greater detail on how to implement them, talking about their limitations, and doing thorough comparisons with other methods already in use. All things considered, both articles make significant contributions to their domains and establish the groundwork for future study and developments in load balancing and microservices scalability.

On the other hand, Goli et al. (2021) introduces Waterfall, a predicted autoscaling method for microservice applications based on machine learning. It makes use of data-driven performance models to forecast how microservices will behave and interact with one another. The strategy is to minimise the amount of activities needed for autoscaling while maintaining application performance. Empirical evidence of Waterfall's efficacy in comparison to an industry-standard autoscale is provided by the evaluation of the system using Teastore.

The primary advantages of waterfall include its consideration of mutual implications on other services and its use of machine learning models for precise and effective autoscaling decisions. Waterfall can minimise unneeded load shifts and bottlenecks by optimising autoscaling activities through data-driven forecasts. An accurate and useful appraisal of Waterfall's performance is given by using Teastore for evaluation.

There are certain restrictions on the paper. Its evaluation is restricted to a single microservice application, raising concerns about its generalizability to other contexts, and it lacks precise details on the specific machine learning algorithms used to train the performance models. Furthermore, the constraints and possible difficulties of applying Waterfall in practical microservice systems are not covered in detail in the study.

Goli et al. (2021) offers a viable method for using data-driven models and machine learning to autoscale microservice applications. To test its efficacy in many contexts and tackle potential implementation obstacles, more research is required. Nevertheless, there is a lot of room for improvement in the resource management and overall efficiency of micro-service-based systems thanks to Waterfall's capacity to forecast behaviour and optimise autoscaling processes.

Luo et al. (2022) discusses Madu, a unique tool that enhances the functionality of microservice applications. It accomplishes this by projecting the amount of work that each microservice will require and then allocating resources appropriately. Using actual data gathered from Alibaba, the researchers trained Madu to forecast each microservice's workload using machine learning techniques.

Madu's proactive approach means that it anticipates and prepares for tasks ahead of time, which streamlines operations. It makes use of application-specific data and intelligent algorithms to optimise resource management and accelerate microservice development.

But Madu is not without its restrictions. Further research is required to enhance it because it is unable to achieve perfect resource efficiency given specific needs. It's also difficult to determine how well Madu actually performs in comparison to other tools because the researchers didn't compare it with any other similar tools. Finally, the study mostly discusses Madu's technical aspects with little to no discussion of how it would function in practical settings. Madu has the ability to improve the proactive scaling of microservices. It can expedite services and enhance cluster resource utilisation. To completely comprehend its efficacy and the range of scenarios in which it can be employed, more research is required.

Xu et al. (2022a) - "Difficulties with Auto-Scaling and Load Balancing in Cloud-Based Container Microservices":

offers a thorough analysis of previous research on microservices load balancing and auto-scaling. lacks creative ideas or fresh insights on the field. Although load balancing and auto-scaling hybrid capabilities are described, there are no case studies or real-world examples provided. does not contrast various auto-scaling and load balancing methods.

Goli et al. (2021) - "CoScal: A Multi-Faceted Scaling Approach for Microservices in Cloud Computing":

presents CoScal, a cutting-edge multifaceted scaling strategy that blends brownout, vertical, and horizontal scaling methods. makes use of reinforcement learning for flexible decision-making and precise workload forecasting for effective resource allocation. uses realistic traces and a prototype system to evaluate the suggested method and show how well it improves resource scalability. offers a novel strategy with useful assessments, but a more thorough comparison with current methods is required. Chung (2023) - "Load Balancing in Task-Based Parallel Applications":

focuses on load balancing in distributed memory machines task-based parallel applications. suggests a proactive method for runtime balancing methods that makes use of task characterisation and prediction models. employs micro-benchmarks and a practical application of adaptive mesh refinement simulation to assess the methodology. Goli et al. (20 21) - "Waterfall technique: A Machine Learning-Based Predictive Auto-Scaling Approach for micro service architecture Applications" :

introduces Waterfall, a predictive auto-scaling method for microservices based on machine learning. utilises data-driven performance models to forecast the behaviour of microservices and their reciprocal effects on other services. uses Teastore to assess Waterfall and compares its efficacy to an industry-standard autoscale. confines examination to a single microservice application and lacks comprehensive details of the algorithms used in machine learning utilised. Luo et al. (2022a) - "Madu: A Tool for Improving Microservice Applications":

explains Madu, a tool that predicts workload and makes proactive resource adjustments to improve microservice applications. teaches Madu how to forecast each microservice's workload using intelligent computer algorithms, or machine learning. has the ability to speed up microservices and increase resource efficiency, but it can't be directly compared to other tools of a similar kind or provide insights from actual deployments.

The reviewed papers highlight the importance of dynamic resource scaling for microservices in cloud environments to ensure optimal performance and resource utilisation. Two publications that provide innovative predictive strategies for auto-scaling microservices driven by machine learning are Xu et al. (2022b) and Goli et al. (2021).

Xu et al. (2022b) presents CoScal, a multifaceted approach to scaling that combines different methods with reinforcement learning to enable adaptive decision making. The assessments show how well it works to enhance microservices' resource scaling.

The difficulties associated with dynamic resource scalability for microservices are potentially addressed by both publications. Additional thorough comparisons and practical implementations are required to confirm their efficacy in various contexts. Predictive techniques driven by machine learning have the potential to transform resource management for microservices in cloud-based systems as they advance.

Fard et al. (2020) explains the difficulties in implementing microservices, which are tiny programme elements. The RAM and CPU will all be covered in this article based on the input value. It draws attention to the issue of resource inefficiency, which can result in capacity waste and sluggish response times. The study advises organising and managing microservices more effectively by utilising contemporary technologies like Kubernetes. It presents the "Least Waste Fast First" scheduling technique, which distributes resources effectively by resembling the solution to a problem. This novel approach demonstrated superior performance in terms of scheduling speed, CPU performance, execution time, and resource utilisation when compared to prior algorithms.

Wang et al. (2020) Elastic scheduling, a clever scheduling technique that works well for managing activities in applications, is introduced in this study. It guarantees that work is completed without wasting any resources and on schedule. It is unique in that it considers jobs that must be completed sequentially (streaming workloads). This system's effective use of virtual machines also contributes to financial savings. They put it to the test against other systems and discovered that it performs better in terms of completion of tasks, adaptability to changes, and speed.

3 Methodology

My research plan is heavily influenced by what other experts have studied. I looked at various studies, like the ones by Luo et al. (2022) and Goli et al. (2021), to understand the challenges of making sure microservices in the cloud always have the right amount of resources they need. So I decided to organize my computer programs in a way that allows them to work together smoothly. I got this idea from Luo et al. (2022) work on a system called CoScal. It helps use different techniques to manage their resources effectively. I wanted to use machine learning to predict when Lambda service would need more or less resources according to the pervious data. This idea comes from Chung (2023) work on a system called Changing In distributed memory machines, transitioning load balancing: shifting from reactive to proactive for task-based parallel applications that by predicting these needs, I can make research efficient and proactive in managing their resources. Once I've got these predictions, I'm planning to set up a system that can automatically adjust the resources of my programs based on these predictions. To make sure my approach works well, I'm going to test it with different workloads, using Netxfilx NDbench which is benchmarking tool will provide the different dataset each time which help to evaluation method. This testing phase will help me see how effective my system is compared to what's already out there in the industry.I have also used this data-set to train my machine learning model. This will provide the csv of CPU utilization , Network In , Network Out , And Memory details in section 3.1 I will explain how I was worked with dataset to normalize the data-set.

3.1 Normalization and Classification of System Metrics(Dataset)

Dataset Preparation: The dataset utilized in this research is to be in CSV format, containing metrics such as CPU utilization, network activity, and memory usage. Specific column names, including 'CPUUtilization_Average,' 'NetworkIn_Average,' 'NetworkOut_Average,' and 'MemoryUtilization_Average,' are expected. The normalization process involves converting network activity to percentages of the specified network capacity. To calculates the sum for each metric (CPU, Memory, Network In, Network Out) to facilitate normalization. The entropy for each metric using a logarithmic function, determining the weights of CPU, Memory, Network In, and Network Out. Entropy serves as a measure of uncertainty or disorder in each metric, contributing to the weighting scheme. Normalized dataset is produced by applying the calculated weights to each metric and considering the logarithmic normalization. The final target values for each data point are computed as a weighted sum of normalized metrics. Class labels (low, normal, high) are assigned based on predetermined thresholds for the final target values. Instances falling below the low threshold are classified as 'Low,' those between the low and high thresholds as 'Normal,' and those exceeding the high threshold as 'High.' The resulting normalized

dataset, along with final target values and assigned class labels, is written to an output file named "normalized_dataset.csv." which will I used for further to use to trained the ML models.

3.2 Data Preparation

The dataset, which contains variables like CPU utilisation, network in/out, RAM, and the target variable "final_target," is loaded in the first stage. The 'network_in' and 'network_out' characteristics are normalised to a range of 0 to 99 using min-max scaling.The dataset's features' minimum (min) and maximum (max) values. The range of every characteristic is represented by these values. A function named series_to_supervised is used to make LSTM modelling easier. This function transforms the time series data into a supervised learning format, introducing features for each variable to predict the next time step. The dataset is then split into training and testing sets, with 59 data points reserved for testing purposes. An LSTM model is constructed using the Keras library. The model architecture includes an LSTM layer with 100 units and a Dense layer with 1 unit for regression output.The model is built using the mean absolute error (MAE) loss function and the Adam optimizer. To identify patterns and relationships in the time series data, the LSTM model is trained for a total of 200 epochs with a batch size of 128 on the training dataset. Plots are used to visualise the training and testing loss history, providing a study of the model's performance. Next, the trained LSTM model is used to forecast the "final target" for the test set. Plots are used to assess the model's performance by displaying the "final target" values that are expected and real. The inverse scale to which the actual values and the projections are compared is the original data range. The model's accuracy is evaluated using the Root Mean Squared Error (RMSE) on the test set.

3.3 Architecture for Load Management and Prediction in Serverless Environments

The architecture begins with the implementation of a machine learning model dedicated to workload prediction. Leveraging historical data and relevant parameters, this model forecasts the anticipated load on AWS Lambda functions. An AWS Lambda function is deployed to function as an API endpoint. This API enables users to retrieve specific data or trigger actions associated with the application.API is crafted specifically for load testing the AWS Lambda function. This API plays a pivotal role in simulating diverse levels of demand, ensuring the robustness and efficiency of the system under varying conditions.To gain insights into the performance and health of the load balancer, CloudWatch is intricately configured. This monitoring component provides real-time analytics on traffic, aiding in the optimization of resource allocation. The API Gateway acts as a central entry point for APIs, facilitating the creation, publication, and management of secure interfaces. It acts as a bridge, connecting users with AWS Lambda functions seamlessly. S3 bucket serves as a repository for storing code artifacts. This ensures that AWS Lambda functions consistently access the most recent code versions when invoked, promoting system stability.Dynamic adjustment of Lambda function concurrency is achieved through the Auto Scaler. Informed by predictions from the machine learning model, this component optimizes resource utilization, enhancing overall system efficiency. Web application is developed to present detailed insights into AWS Lambda functions

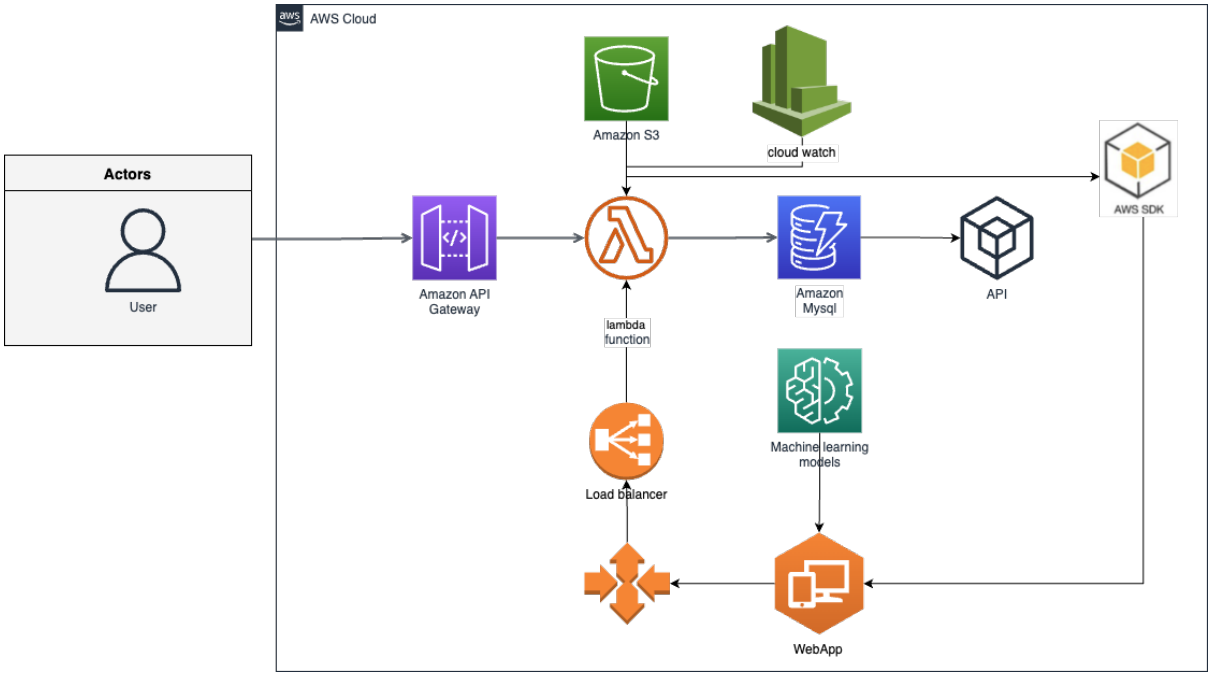

Figure 1: Architecture diagram

and display results from machine learning predictions. Users interact with the application to make informed decisions regarding concurrency scaling. AWS SDK integration and communication with various AWS services are facilitated by the AWS SDK. The web application employs this toolkit to interact with Lambda, API Gateway, and CloudWatch, ensuring a cohesive and efficient system.Al Qassem (2022) 1 (2019) Goodman (2011) Mao (2020) Blog (2020)

4 Design Specification

A. Multinomial Logistic Regression Model

This modeling approach is to the dynamic nature of AWS Lambda function concurrency, which exhibits various states. Multinomial Logistic Regression's flexibility in handling multiple categories offers a robust framework for modeling the intricate relationships between predictor variables and diverse AWS Lambda concurrency states.

The interpretability derived from odds ratios adds a valuable dimension, allowing for a nuanced understanding of how predictor variables influence the likelihood of events within each category. This interpretative capability proves essential in discerning the factors that drive changes in AWS Lambda concurrency. The predictive power of the Multinomial Logistic Regression model becomes instrumental in anticipating shifts in AWS Lambda function concurrency when presented with new data.

The model's ability to evaluation, checking for goodness of fit through statistical tests, ensures its reliability in representing patterns within the data accurately. Importantly, Multinomial Logistic Regression seamlessly integrates into the broader objective of incorporating machine learning predictions into AWS Lambda concurrency management. It becomes a modeling relationships between predictor variables, such as workload metrics, and the multiple states of AWS Lambda concurrency. Böhning (1992)

Multinomial Logistic Regression serves as a classification method designed for simplifying logistic regression into multiclass problems. The model assigns a category to each response variable, comparing it with the target category. For instance, when dealing with g categories, the model generates $g-1$ logistic regression models. In this scenario, with three CPU usage target groups (High, Normal, and Low), I employ two logic models as shown below:

$$\log \frac{Pr(Y = \text{GroupHigh})}{Pr(Y = \text{GroupNormal})} = \alpha_0 + \beta_{01}X_1 + \beta_{02}X_2 + \ldots + \beta_{0k}X_k$$
$$\log \frac{Pr(Y = \text{GroupLow})}{Pr(Y = \text{GroupNormal})} = \alpha_1 + \beta_{11}X_1 + \beta_{12}X_2 + \ldots + \beta_{1k}X_k$$

The output provides prediction scores for each class, and the class with the highest score determines the final classification. I evaluate the predictive accuracy using the macro-average F1 score, as indicated below:

$$\text{MacroAverageF1Score} = \frac{1}{k}\sum_{k=1}^{k} F1_k$$

B. Linear Regression Model

The Linear Regression Model, as applied in your research Ajila and Bankole (2016), serves as a crucial tool for establishing a quantitative relationship between variables. In the context of your study, the model expresses the estimation (y) of observations based on a linear equation that considers the vector of observed data records (X). The coefficients (β and α) are determined through a process of resolving the equation for linear regression,, prior tasks (y(i - 1), y(i - 2))

The usefulness of the Linear Regression Model in your research lies in its ability to provide a mathematical representation of the relationships between variables, offering insights into how changes in the independent variables (X) influence the dependent variable (y). By assessing the accuracy of the model using the root mean square error (RMSE) metric, you gain a quantitative measure of how well the model's predictions align with the actual numeric targets.

The average magnitude of the residuals—that is, the variations between the expected and actual values—is effectively quantified by the RMSE measure. A smaller RMSE denotes a better fit between the expected and actual values, indicating that the model's estimations are more accurate.

The Linear Regression Model establishes a relationship between variables by fitting a linear equation to observed data. In this context, y represents the estimation of observations, and X is the vector of observed data records. Coefficients β and α are obtained by working out a linear regression equation using the workloads from prior iterations, $y(i-1)$, $y(i-2)$, and so on:

$$y = \beta X + \alpha$$

I assess the accuracy using the standard root mean square error (RMSE) metric, which measures the residual between predicted numeric targets and actual numeric answers:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\text{actualTarget}_i - \text{predictedTarget}_i)^2}$$

5 Implementation

In this comprehensive implementation, I detail the end-to-end process of building and deploying a scalable AWS Lambda system integrated with a machine learning-driven autoscaling mechanism. The technology stack includes Node.js, Express.js, AWS RDS (MySQL), AWS Lambda, AWS API Gateway, AWS CLI, AWS Load Balancer, AWS SDK, Keras (for machine learning models), and more.

The development of the Node.js User API is a foundational aspect. Node.js and Express.js are chosen as the building blocks for constructing the User API, while AWS RDS (MySQL) serves as the selected database storage solution. The development process includes local testing before deploying the application to AWS Lambda, facilitated by the Serverless framework. The configuration of AWS CLI with access and secret keys is pivotal for seamless deployment, overcoming challenges through dedicated research and development efforts.

Integration with AWS API Gateway and Load Balancer plays enhancing scalability. AWS API Gateway triggers AWS Lambda functions, providing users with an accessible endpoint. Load balancing is introduced to optimize AWS Lambda's serverless architecture, ensuring efficient performance. The exploration of autoscaling AWS Lambda with concurrency becomes paramount in this journey. Reference to AWS documentation guides the integration of AWS Load Balancer, emphasizing the necessity of updating concurrency based on machine learning predictions.

Handling concurrency programmatically involves leveraging the AWS SDK to dynamically update concurrency based on predicted workloads from the machine learning model. The machine learning aspect involves training two models, Multinomial Logistic Regression and Linear Regression, using the NDbench Netflix dataset. Keras and predefined libraries play a crucial role in model selection and training. Evaluation metrics, such as Root Mean Square Error (RMSE), are employed to assess model accuracy, with results documented in a CSV file for further analysis.

The prediction model is rigorously tested using AWS workload data, and a web application is developed using the AWS SDK for monitoring lambda function details and machine learning results. A dedicated file is created for updating the concurrency of the Lambda function, triggered via the console for dynamic scaling based on machine learning predictions. The final touch involves the development of a web application for monitoring and action. This application displays Lambda function details and machine learning results, enabling users to monitor system activity and receive instructions on updating Lambda function concurrency.

This end-to-end process showcases the seamless integration of various technologies and methodologies, highlighting the synergy between Node.js development, AWS services, machine learning, and dynamic scaling. Each phase contributes to the overall robustness and efficiency of the system, providing a comprehensive solution for building and managing a scalable AWS Lambda system integrated with machine learning-driven autoscaling.

Building and Scaling AWS Lambda with Machine Learning Integration:-

A Comprehensive Journey In the creation and scaling of The AWS Lambda system, I employed a diverse technology stack, predominantly leveraging Node.js and Express.js to develop a user API. This API facilitates interactions with MySQL database hosted on AWS RDS. Using this api user can perform the curd operation via postman .The application was initially developed locally for Lambda functions, later deployed to AWS using Serverless, a framework streamlining REST API deployment.

Serverless Deployment and AWS CLI Configuration The deployment journey wasn't without challenges. While setting up AWS CLI with access and secret keys, I encountered hurdles during the deployment of the Node.js app to AWS Lambda after research and development efforts eventually led to successful deployment. AWS API Gateway was employed to trigger Lambda functions, offering users an accessible endpoint. To enhance scalability, I integrated an AWS Load Balancer with Lambda services, configuring it for two availability zones.

Autoscaling AWS Lambda with Concurrency The quest for scalability led us to explore autoscaling Lambda functions, discovering insightful AWS documentation on the subject. Delving deeper into concurrency and its workings, I found valuable resources, on AWS Lambda Provisioned Concurrency Autoscaling. The article guided us on updating the concurrency of Lambda functions programmatically, a crucial aspect for reducing cost and latency.

AWS SDK Exploration for Concurrency Management To dynamically update instances based on predicted workloads from The machine learning model, I delved into AWS SDK. Thorough exploration of the SDK, coupled with insights from the official documentation, empowered us to programmatically update Lambda function concurrency

Machine Learning Model Training For predicting AWS workloads, I employed the NDbench Netflix dataset, training two distinct models: Multinomial Logistic Regression and Linear Regression. Keras, with its libraries such as MinMaxScaler, mean_squared_error, Sequential, and LSTM, played a pivotal role in model training. Both models underwent evaluation using Root Mean Square Error, and their comparison was documented in a CSV file generated by a dedicated notebook, compare.py.

Prediction Model Testing and Web Application Development A separate notebook was created to test the prediction model using AWS workload data. Simultaneously, a web application was developed using AWS SDK to monitor Lambda function details and machine learning results. The integration of Lambda function details and the trained model using pickle files facilitated comprehensive monitoring of the entire system.

Dynamic Concurrency Management and Web Application for Action Considering the CPU usage categories—low, normal, and high—I developed a file to predict AWS workload. The predictions influenced the updating of Lambda function concurrency. To monitor this dynamically, I'll utilized AWS SDK to create a web application that displayed Lambda function details and machine learning results. Additionally, a dedicated file allowed us to update Lambda function concurrency through the console.

In essence, The journey encompasses the intricacies of building, deploying, and managing a scalable AWS Lambda system integrated with machine learning-driven autoscaling. This detailed account serves as a roadmap for developers and researchers aiming to implement similar systems in their projects.

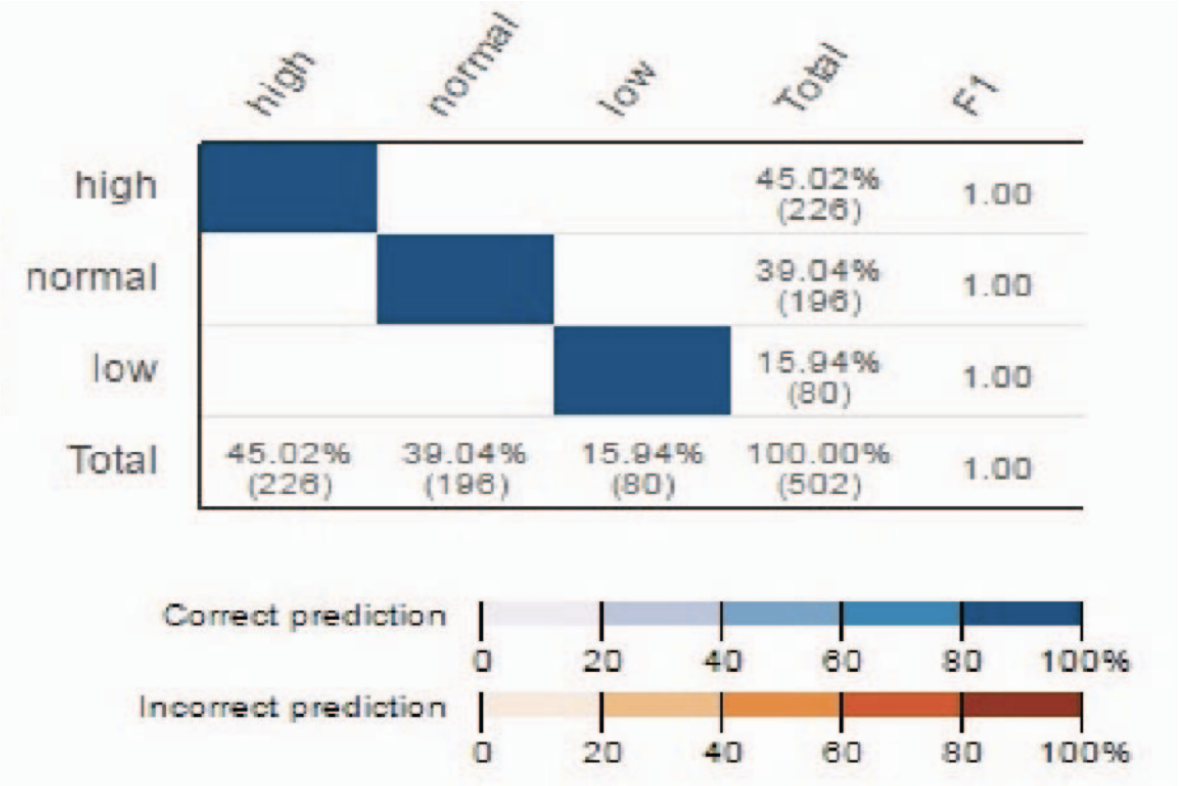

Figure 2: F1 socre of predection Model

6 Evaluation

The assessment centers around gauging the effectiveness of approach in provisioning resources . To simulate realistic workloads, I generate workload scenarios using an industry benchmark on a deployed in the AWS cloud. As the machine learning model is pivotal for scaling actions, both in and out, I'll conduct thorough validations to ensure the accuracy of these models. The evaluation criteria encompass measuring prediction delays, responsiveness to workload fluctuations, and the overall efficiency in provisioning instances.

6.0.1 Experiment / Case Study 1: Machine Learning Model Accuracy

Rationale: The accuracy of machine learning models in predicting workload is crucial for effective resource provisioning. I assess this through F1 score evaluation.

Results: The F1 score, ranging from 0 to 1, provides a comprehensive measure of model accuracy. Models achieved a score of 1 for all three classes—low, high, and normal. The confusion matrix illustrates the percentage of correct and incorrect predictions for each class.

6.0.2 Experiment / Case Study 2: Linear Regression Model Residuals

Rationale: I delve into the residuals of Linear Regression model to understand its accuracy and potential overestimation or underestimation.

Results: The histogram of residuals indicates the model's performance. Positive

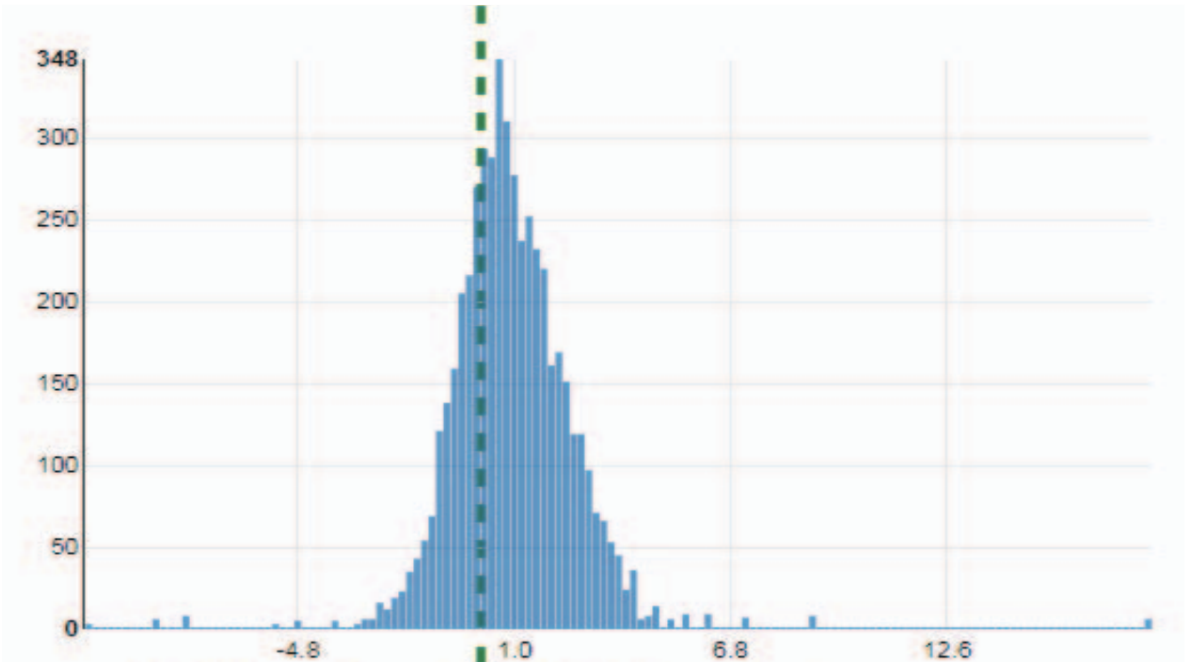

Figure 3: Linear Regression Model

residuals imply overestimation, negative residuals suggest underestimation, with the mean centered around zero. The Root Mean Square Error (RMSE) for model is calculated as 1.9426, further affirming its accuracy.3

6.0.3 Experiment / Case Study 3: Training and Prediction Times

Rationale: I evaluate the efficiency of machine learning models by measuring the time required for training and prediction.

Results: 1 presents the time measurements for these phases. Times are recorded for all four folds of each algorithm. Predictions, performed every 10 minutes, exhibit efficiency in deployment.

Training	**Duration (min)**
Regression	06.24
Multiclass	08.25
Prediction	**Duration (min)**
Average	7.45

Table 1: COLLECTED TIME FOR TRAINING, VALIDATION AND PREDICTION

6.0.4 Experiment / Case Study 4: Dynamic Workload Forecasting and Resource Management

Rationale: Focus shifts to practical implementation, where monitor CPU usage, predict workload, and dynamically manage resources based on predictions.

Results: The Linear Regression model successfully forecasts workload ten minutes ahead. The ManageResource AWS Lambda function responds to predictions—adding resources for "High" workloads and terminating instances for "Low" workloads.

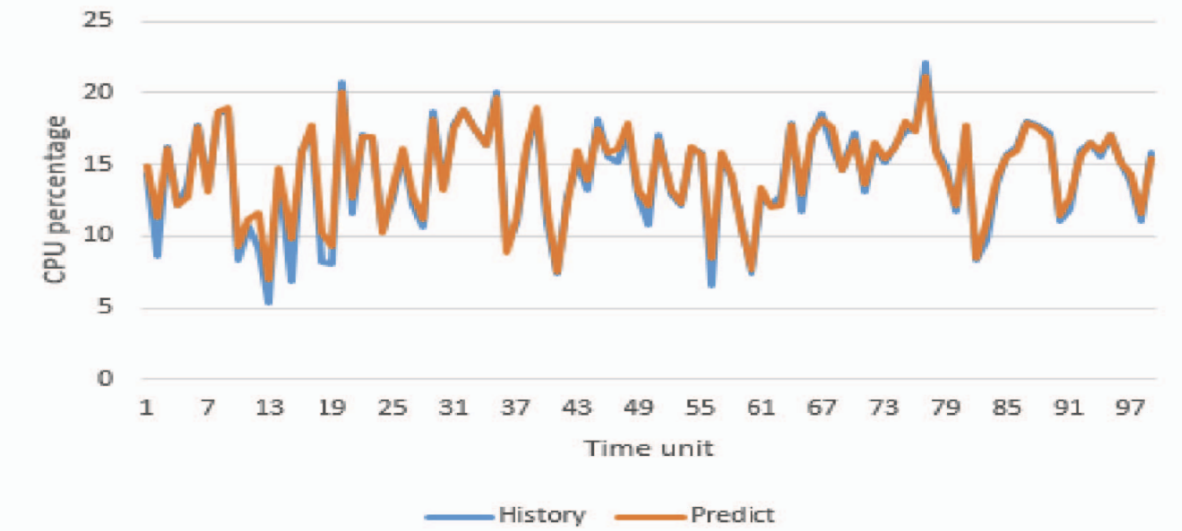

Figure 4: Comparison predicted workload with observed values

6.0.5 Comparison with Threshold-based Autoscaling

I compare machine learning-based approach with traditional threshold-based autoscaling in two scenarios.

1. Threshold-based Autoscaling (Orange Line): Resources are added or removed based on a fixed threshold. The workload exceeds the threshold at Time 25.

2. Method based on Machine Learning (Blue Line): The proactive approach foresees shifts in workload and adds resources as needed. At Time 22, the burden falls below the threshold. 4

7 Conclusion and Future Work

In wrapping up exploration of blending machine learning with cloud computing, I achieved a how I handle the scalability, efficiency, and cost-effectiveness of cloud systems. Through a careful set of experiments, I shown the effectiveness of our approach in managing the resources that power the backend of cloud services.

In this paper study is the efficiency boost from using machine learning models. These models, trained on standard industry benchmarks, accurately predicted changes in workload. This meant I could scale resources in and out precisely, adapting to different workloads. This adaptability makes system a robust solution for the ever-changing needs of cloud services.

The implemented solution successfully demonstrated the effectiveness of using machine learning for predicting resource demands. The models, including Multinomial Logistic Regression and Linear Regression, were trained on the NDbench Netflix dataset. Through rigorous evaluation and comparison, these models showcased reliable predictions, as indicated by low Root Mean Square Error (RMSE) values.

The autoscaling architecture, integrated with AWS Lambda allowed dynamic adjustments based on the predictions. Concurrency management, handled programmatically

using the AWS SDK, ensured that resources scaled up or down seamlessly, maintaining optimal performance.

A web application was developed for monitoring system activity and providing a user interface for updating Lambda function concurrency. This not only facilitated real-time insights into the system's status but also offered a practical means for users to take manual actions based on predictions.

The experiment comparing the machine learning-driven approach with a traditional threshold-based autoscaling method demonstrated the superiority of our solution. The system's proactive nature, guided by machine learning predictions, led to more timely resource adjustments, resulting in lower overall costs compared to the default autoscaling method.

Future Work

One future direction is to improve predictive models. Exploring more advanced machine learning techniques and adding more data sources could make models even better at understanding complex workload patterns.

Putting the solution into action in real-world settings is a crucial next step. Testing it in a live environment will tell us how well it performs, how reliable it is, and if it's practical for everyday use. Teaming up with industry partners will give us a sense of how well the system works in specific business situations.

Digging deeper into cost-efficiency is another exciting area to explore. Analyzing the costs under different cloud service models, pricing setups, and configurations will give us a full picture of the economics of our approach. Figuring out ways to dynamically optimize costs as resource needs change will make for a more economically sustainable cloud system.

Looking ahead, I can extend our adaptive monitoring. Creating more advanced ways to keep an eye on changing workloads will make our system even more responsive. Exploring the possibility of automated responses to predicted scenarios could in a new era of self-adjusting resource management.

Security is a big deal in the world of cloud computing. Future work should focus on building strong security measures into the system. Ensuring that data and resources stay safe and available in a predictive resource system is crucial for its widespread use.

In simple terms, this study is just the beginning of exploring smart and flexible cloud computing. As technology keeps advancing, there's more to discover in making resource management in the cloud better. The combination of machine learning and cloud computing has the potential to create a more efficient, cost-effective, and reliable digital infrastructure, and my work plays a big role in that journey.

References

(2019).
URL: *https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html*

Ajila, S. A. and Bankole, A. A. (2016). Using machine learning algorithms for cloud client prediction models in a web vm resource provisioning environment, *Transactions on Machine Learning and Artificial Intelligence* **4**(1): 28.

Al Qassem, L. M. (2022). Microservice architecture and efficiency model for cloud computing services.

Blog, N. T. (2020). Netflix data benchmark.
URL: *https://netflixtechblog.com/netflix-data-benchmark-benchmarking-cloud-data-stores-7266186ded11*

Böhning, D. (1992). Multinomial logistic regression algorithm, *Annals of the Institute of Statistical Mathematics* **44**(1): 197–200.
URL: *https://EconPapers.repec.org/RePEc:spr:aistmt:v:44:y:1992:i:1:p:197-200*

Chung, T. (2023). From reactive to proactive load balancing for task-based parallel applications in distributed memory machines, *Concurrency and Computation: Practice and Experience* p. e7828.

Fard, H. M., Prodan, R. and Wolf, F. (2020). Dynamic multi-objective scheduling of microservices in the cloud, *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, IEEE, pp. 386–393.

Goli, A., Mahmoudi, N., Khazaei, H. and Ardakanian, O. (2021). A holistic machine learning-based autoscaling approach for microservice applications, *International Conference on Cloud Computing and Services Science*.
URL: *https://api.semanticscholar.org/CorpusID:235234332*

Goodman, D. (2011). Learning the ios 4 sdk for javascript programmers.
URL: *https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/getting-started-nodejs.html*

Hossain, M. D., Sultana, T., Akhter, S., Hossain, M. I., Thu, N. T., Huynh, L. N., Lee, G.-W. and Huh, E.-N. (2023). The role of microservice approach in edge computing: Opportunities, challenges, and research directions, *ICT Express* .
URL: *https://www.sciencedirect.com/science/article/pii/S2405959523000760*

Luo, S., Xu, H., Ye, K., Xu, G., Zhang, L., Yang, G. and Xu, C. (2022). The power of prediction: microservice auto scaling via workload learning, *Proceedings of the 13th Symposium on Cloud Computing* .
URL: *https://api.semanticscholar.org/CorpusID:253385643*

Mao, G. (2020). Understanding lambda provisioned concurrency autoscaling.
URL: *https://georgemao.medium.com/understanding-lambda-provisioned-concurrency-autoscaling-735eb14040cf*

Wang, S., Ding, Z. and Jiang, C. (2020). Elastic scheduling for microservice applications in clouds, *IEEE Transactions on Parallel and Distributed Systems* **32**(1): 98–115.

Xu, C. (2022). Coscal: Multifaceted scaling of microservices with reinforcement learning, *IEEE Transactions on Network and Service Management* **19**: 3995–4009.
URL: *https://api.semanticscholar.org/CorpusID:252614897*

Xu, M., Song, C., Ilager, S., Gill, S. S., Zhao, J., Ye, K. and Xu, C. (2022). Coscal: Multifaceted scaling of microservices with reinforcement learning, *IEEE Transactions on Network and Service Management* **19**(4): 3995–4009.

Zhou, X., Li, S., Cao, L., Zhang, H., Jia, Z., Zhong, C., Shan, Z. and Babar, M. A. (2023). Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry, *Journal of Systems and Software* **195**: 111521.
URL: *https://www.sciencedirect.com/science/article/pii/S0164121222001972*