Rule-based Task Scheduling in Cloud-Edge Computing for Energy and Resource Utilization Optimization

Jialing Chen
x23158131

Abstract

Smart devices and real-time services are becoming more common nowadays, which shows problems with using only cloud systems. These problems include systems getting too busy. Because of these issues, there is a need to find better ways to use both edge devices and cloud computing together to handle tasks more efficiently. The main goal of this project was to create a basic system that helps decide where to run different tasks - either on edge devices or in the cloud. This decision depends on how busy the system is at any given time, looking at things like how much CPU and memory are being used. The project used a rule-based system running in Kubernetes to make these decisions.

To keep track of how the system was doing, Prometheus was used to watch different measurements in real-time. The system used these measurements to figure out where tasks should go based on some fixed rules that were set up at the start. Different tests were done to check if this basic rule setup actually worked well. Looking at the test results showed that the fixed rules did a decent job handling different amounts of work coming into the system. Even though the rules stayed the same and couldn't change on their own, the system still managed to run tasks faster and use less power than when no rules were used at all. This shows that sometimes having just a few simple, unchanging rules can actually make things work better, especially when dealing with both cloud and edge computing together.

What's interesting is that this system worked well without using complicated machine learning. This shows that sometimes simpler approaches using basic rules can be really effective, especially for smaller edge devices. The results suggest this kind of system could be useful in real situations where both edge and cloud computing are needed.

Keywords: Edge Computing, Cloud Computing, Rule-based Scheduler, Resource Management, Energy Efficiency, Kubernetes

1 Introduction

1.1 Background

Smart devices and real-time services are now part of everyday life, making quick system resource and energy management essential. When dealing with lots of data, regular cloud systems often struggle to keep up. These systems can get too busy, which leads to wastes

energy and resoucer distribution imbalance. When systems get really busy, they start to use up more resources and overload the energy. To fix these problems, researchers started using cloud-edge computing (Shi et al., 2016). This approach puts some tasks closer to where the data comes from, which helps make the system work better overall.

Work gets split between powerful computers in the cloud and edge closer to where people use them. Big tasks that need lots of processing power happen in the cloud, while edge devices handle quick, local jobs. While this setup helps, picking the right place for each task remains tricky. Current systems need clear rules to decide where each task should run. Without these steps, computers might sit idle while others are too busy, which uses extra electricity and makes users wait longer for their results (Pan et al., 2018).

One big problem is keeping track of what's happening in the system - like when processors get overloaded, memory starts running low, or too many people try to use the system at once. Setting up basic rules helps control these situations. For example, if the system sees the processor getting too hot, it can move some work somewhere else. These kinds of simple instructions work well for everyday problems. When the rules are easy to understand and follow, the whole system runs more smoothly. Even basic scheduling instructions can make good choices about which computer should handle each task, helping to save energy and get work done faster (Tuli et al., 2020; Wang et al., 2021).

As more computers and devices join the network, it becomes harder to keep track of all the work that needs to be done. Simple steps tell the system where to send each piece of work. Making these steps work well is hard because some computers are big and powerful, while others are small and basic, but they all need to work together (Mao et al., 2017).

Looking at how computers share work between cloud and edge systems, three main issues keep coming up (Liu et al., 2023). Moving work from cloud to edge takes time and can make everything slower. Trying to use less electricity often means the system runs too slowly to be useful. Also, when many people use the system at once, it has trouble keeping up with all the requests (Chen et al., 2018). While some solutions get really complicated, using basic rules for organizing work might actually be better for everyday use. The real trick is keeping the system both useful and easy to work with.

1.2 Motivation

More and more devices connect to networks each day, creating new challenges for processing their information. These devices need their data handled quickly to work properly. Sending everything to cloud servers often causes delays, especially when lots of people are using the system. This slowdown makes services work poorly and burns through more power than it should

The challenge in current systems lies in setting up effective rules for task distribution. Without proper scheduling rules, tasks might end up in the wrong place - either overloading the cloud servers or putting too much strain on edge devices. A good set of scheduling rules needs to consider things like how powerful each device is, how much energy it uses, and where the data is coming from. Getting these rules right can help stop the cloud from getting too busy while making sure edge devices are used properly and don't overheat.

What makes this really interesting is that many existing solutions try to use complic-

ated methods like machine learning to solve these problems. But for lots of real-world situations, especially when dealing with smaller edge devices that don't have much processing power, simpler might actually be better. A straightforward set of well-thought-out rules could work just as well, while being easier to set up and fix when something goes wrong. Plus, when the rules are simple and clear, it's easier to understand why the system makes certain decisions, which helps a lot when trying to make sure everything runs smoothly and efficiently.

1.3 Problem Statement

Most computing systems today need better ways to handle task scheduling between cloud and edge devices. The system needs clear steps to pick the best place for each job to run. Each choice depends on checking processor load, free memory space, and how busy the network is. Without proper steps, work ends up in the wrong places, making everything slower and wasting electricity. Edge devices make this harder because they have less power than cloud servers, so their workload needs careful planning.

The system works best with simple steps for handling each task. Some people try to use smart computer programs to make decisions, but these programs need too much processing power and don't work well on edge devices. What's needed are basic instructions that can spread out work evenly across different computers. These instructions should help everything run faster and save power, without using complex technology.

Research points to three common problems in cloud-edge systems (Liu et al., 2023). First, tasks take extra time when moving between cloud and edge servers. Second, efforts to cut power use often slow down the whole system. Third, the system slows down when too many people try to use it at once (Chen et al., 2018). Instead of using complex fixes, basic rules for handling tasks often work better for everyday needs. The goal is to make something that works well but stays simple enough to use in real life.

1.4 Research Question

"What basic steps can help save power and make resource management better when handling everyday tasks in cloud-edge computing environments?"

1.5 Problem Solution

Making cloud-edge systems work better doesn't need complicated programs. Simple checks can tell how the system is doing right now - like seeing if processors are too busy, if there's enough free memory, or if data is moving smoothly through the network. These basic checks help decide whether work should stay on edge devices or go to the cloud (Tuli et al., 2020).

Tools like Prometheus check how each part of the system is working. They measure which computers are busy and which ones have free time. When one computer gets overloaded while others sit empty, basic rules help spread out the work more evenly (Wang et al., 2021). Other studies have looked at similar ideas, but this way focuses on being simple and easy to use (Perin et al., 2022).

Tests show that basic scheduling rules work well in everyday use (Zhang et al., 2022). Unlike complex computer programs, these rules are easier to understand and fix when problems happen. The system runs without needing special equipment or extra processing

power. This works great for edge devices that have limited computing abilities (Kumar et al., 2015). Building on what others have found, this project shows how simple instructions can handle tasks well while keeping everything easy to manage (Shi et al., 2016).

1.6 Objectives

This study looks at a different way to manage work between cloud and edge systems. By following basic steps instead of complex rules, the system tries to save power and optimize the resource management (Perin et al., 2022). The results show that easy-to-follow instructions can do the job just as well as fancy computer programs.

The main tasks for this study include:

1. Create simple steps for checking processor use, memory space, and network traffic.
2. Set up a test system with Kubernetes to move work between cloud and edge.
3. Use monitoring tools like Prometheus to see how everything runs.
4. Run different sized workloads to check how well the steps work.
5. See if these basic steps work better than systems without any special rules (Pan and McElhannon, 2018)

These goals help create a system that's easy to use and fix. Rather than making things complicated with changing rules, basic steps can handle work between cloud and edge just as well (Shi et al., 2016). This simple way works better for small companies that need their systems to run smoothly without dealing with complex programs.

1.7 Limitations and Challenges

Some limits of this study need to be mentioned. Using basic rules to make choices keeps things simple, but might not handle unusual problems well. Smart computer programs could work better in tricky situations, but testing those programs wasn't part of this study. The focus stayed on seeing how well simple steps could handle everyday tasks.

Checking how well the system runs brings up some problems. The monitoring tools need time to gather and show new information. When these updates take too long, it can affect how the system makes decisions about where to send work. Also, if the monitoring tools don't work perfectly, the system might not have all the information it needs to make the best decisions about where tasks should go (Kumar et al., 2015).

Testing brings up another set of limitations. Most of the testing happened using computer simulations rather than real equipment (Pan and McElhannon, 2018). While these simulations help figure out if things work, they might not show exactly how the system would behave in the real world. Things like slow networks, hardware problems, or lots of users requesting things at the same time could cause different issues than what was seen in the test environment (Perin et al., 2022).

The fixed rules themselves have some limitations too. Once they're set up, they stay the same no matter what happens in the system. The basic steps make the system run reliably, which helps in most cases (Shi et al., 2016). While these steps might not be perfect for every single situation, they work well enough for common everyday tasks. The key is setting up rules that handle normal workloads effectively.

1.8 Document Structure

The remainder of this paper is organized as follows:

Section 2 (Related Work) make the review of past years related topic papers about task scheduling in the cloud and edge environment and classify them into AI-based and non-AI methods.

Section 3 (Methodology) describe the whole experiments methods used to test the ruled-based scheduling system.

Section 4 (Design Specification) describe the whole experiment system architecture design and make the explain of how each components work together.

Section 5 (Implementation) describe the whole experiments implementations details, including each implementation steps for each rule.

Section 6 (Evaluations) make the explanation of metrics between baseline and scheduler and compare metrics result for each experiment rule.

Section 7 (Conclusion and Future Work) make the conclusion and the whole experiment and discuss further work to improve this system.

2 Related Work

Studies about managing work between cloud and edge show many different solutions. Some researchers use complex computer programs, while others prefer basic steps. A review of fifteen key studies shows two main approaches. The first group uses smart programs that learn from experience, while the second group uses simple rules and calculations. Each method tries to improve how tasks are handled - some focus on using less power, others on making things run faster, and some on handling many requests at once.

2.1 AI/ML-Based Methods: Reinforcement Learning

TF-DDRL (Wang et al., 2025) The system in this paper mixes two types of smart programs to handle tasks between cloud and edge devices. One part looks at how different tasks connect to each other, while another part learns from experience to make better choices. Tests showed this method helped tasks run faster than older ways of doing things.

A3C with RNN (Tuli et al., 2020) This study built a system that learns from past work patterns to handle tasks better. Their program watches how busy the system gets and adjusts its choices based on what it sees. Tests with different types of work showed it could spread tasks around well, even when things kept changing.

DRL Scheduling (Wang et al., 2024) The study looks at a smart program that learns where to send tasks in cloud and edge systems. After training, the program gets better at choosing the right place for each task. Tests under different conditions showed that this method helped make everything run more smoothly and quickly compared to older ways of doing things.

Hierarchical RL (Zhou et al., 2023) The study created a two-part system for handling tasks in edge computing. One part picks which computer should do the work, while another part decides how much resources to give it. This setup helps get work done faster and handles changes in the system better than simpler methods.

DQN Vehicular Edge (Tang et al., 2020) The study shows a way to handle tasks for cars using edge computing. Their program helps save power while keeping response times quick. It works well even when cars move around and network connections keep changing.

Actor-Critic RL (Al-Turjman et al., 2021) The study tested a new way to manage tasks in smart city systems. The program checks things like power use and network speed before deciding where to send work. Tests showed it helps city services run more reliably and respond quickly to requests.

Multi-Agent DRL (Chen et al., 2022) The study uses several small programs working together to handle tasks between cloud and edge devices. These programs learn from each other to make better choices about where work should go. Tests showed this method helps get tasks done faster and uses computer resources better, even when network conditions keep changing.

RL Resource Allocation (Sharma et al., 2020) The research looks at a program that learns how to share computer resources between cloud and edge systems. It watches how busy things get and adjusts its choices to avoid turning away tasks. Tests showed it helps save power and makes everything run faster across different types of networks.

Blockchain Scheduling (Liang et al., 2022 – partial AI use) The study combines blockchain with task management to make edge computing safer and fairer. This setup helps keep data secure while sending tasks to the right places, without needing one central computer to control everything. Tests in networks with many connected devices showed it helps things run faster.

2.2 Other AI/ML

Federated Privacy Scheduling (Yang et al., 2022) The study shows how edge devices can work together to handle tasks better while keeping user information private. Different devices learn from their own data without sharing it with others. This helps keep everything running quickly while protecting private information.

GNN for Collaborative Edge (Zhou et al., 2021) The study uses special programs that understand how different computers connect to each other in the network. This helps find better ways to send tasks between cloud and edge devices. Tests showed it works well in big networks and helps get work done faster than older methods.

2.3 Non-AI Methods: Metaheuristic / Optimization Algorithms

Genetic Algorithm (Khan et al., 2021) The study looks at a way to save power when handling tasks in edge computing. The program tries to find the best balance

between getting work done quickly and using less electricity. This method works well for city systems and healthcare services, keeping response times good while reducing power use.

Bi-Level Metaheuristics (Zhou et al., 2021) The study created a two-layer system for managing tasks between cloud and edge devices. One layer looks at how the whole system runs, while another layer focuses on making each computer work better. This setup helps spread out work evenly and save power when many devices are connected.

2.4 Non-AI Methods: Architecture / Rule-Based Systems

ENTS: Edge-Native Scheduler (Zhang et al., 2022) ENTS works as a system for edge devices to share tasks without always needing help from the cloud. Edge devices can talk directly to each other to decide who does what work. Tests in places like smart cities showed this helps prevent task failures and makes everything run more smoothly.

EASE: Energy-Aware Scheduling (Perin et al., 2022) EASE helps manage work for vehicles using edge computing and green power. The system checks battery levels, where vehicles are moving, and when tasks need to be done. Tests showed it uses less power while still getting everything done on time, making it good for vehicles that need to save energy.

Table 1: Summary of 15 reviewed papers on rule-based and learning-based task scheduling in cloud-edge computing.

Author	**Title**	**Focus Problem**	**Research Method**	**Solution**	**Findings**
Wang et al. (2025)	TF-DDRL: A transformer-enhanced distributed DRL technique	IoT task scheduling in edge and cloud	Transformer + Distributed DRL	Transformer-enhanced distributed DRL	Improved latency and task handling
Tuli et al. (2020)	Dynamic scheduling using a3c learning and residual recurrent neural networks	Dynamic edge–cloud task scheduling	A3C + Residual RNN	Adaptive scheduling using A3C with RNN	Effective under changing workloads
Wang et al. (2024)	Deep reinforcement learning-based scheduling	Load balancing and response time in edge–fog	Deep Reinforcement Learning	DRL-based scheduler	Improved system stability and delay

Author	Title	Focus Problem	Research Method	Solution	Findings
Zhang et al. (2022)	Ents: An edge-native task scheduling system for collaborative edge computing	Collaborative edge task management	Edge-native Scheduling	Decentralized task scheduler	Reduced failures and improved performance
Perin et al. (2022)	EASE: Energy-aware job scheduling for vehicular edge networks	Job scheduling for vehicular edge networks	Energy-aware Scheduling	Energy-aware task scheduler	Saved energy while meeting deadlines
Zhou et al. (2023)	Hierarchical Multi-Agent Deep Reinforcement Learning	Hybrid computation offloading in MEC	Hierarchical Multi-Agent DRL	Two-level DRL scheduler	Better offloading, reduced delays
Yang et al. (2022)	Federated task scheduling	Privacy-preserving scheduling in fog computing	Federated Learning	Federated learning-based scheduler	Maintained privacy with low latency
Tang et al. (2020)	Deep reinforcement learning for task scheduling in vehicular edge computing systems	Task scheduling in vehicular edge computing	Deep Q-Network	DQN-based scheduler	Reduced energy and maintained low latency
Zhou et al. (2021)	Edge intelligence: Paving the last mile of artificial intelligence with edge computing	Collaborative edge–cloud scheduling	Graph Neural Networks	GNN-based scheduling	Better throughput and latency in complex networks
Al–Turjman et al. (2021)	An overview of security and privacy in smart cities' IoT communications	Smart city task offloading	Actor–Critic RL	RL-based offloading	Improved real-time reliability

Author	Title	Focus Problem	Research Method	Solution	Findings
Chen et al. (2022)	Multi-agent deep reinforcement learning-based task scheduling	Task scheduling in IoT-based edge–cloud systems	Multi-Agent DRL	Multi-agent DRL for collaborative scheduling	Reduced delay and improved resource utilization
Khan et al. (2021)	Edge computing: A survey of technologies, use cases, challenges and research directions	Energy-efficient task scheduling in fog computing	Genetic Algorithm (GA)	GA-based scheduler balancing energy and delay	Lower energy use with acceptable delays
Sharma et al. (2020)	Reinforcement learning-based resource management in smart cities	Dynamic resource allocation in edge–cloud	Reinforcement Learning (RL)	RL-based adaptive resource assignment	Improved energy efficiency and faster response
Liang et al. (2022)	Blockchain-based task scheduling	Secure and fair task offloading in edge computing	Blockchain + Scheduling Algorithm	Blockchain-enhanced task scheduler	Reduced latency, improved security and fairness
Zhou et al. (2021)	A bi-level scheduling model for cloud-edge computing	Performance and energy trade-off in cloud–edge scheduling	Bi-Level Optimization + Metaheuristics	Bi-level optimized scheduler	Balanced energy and workload in dense IoT networks

2.5 Critical Analysis

After reviewing these papers, it is clear that many recent studies focus on advanced AI-based methods, such as Deep Reinforcement Learning (DRL), A3C, or Transformer models. These techniques show strong results in some experiments, for example in Wang et al. (2025) and Tuli et al. (2020), where they improved performance and reduced response time. However, the main problem is that they are very complex and need a lot of training data and computing power. In edge environments, where resources are limited, this is difficult to apply in practice.

Some works, like EASE or ENTS, introduced rule-based methods. These are simple, lightweight, and use less energy. But the limitation is that they are fixed scheduling approaches. They do not have learning ability and cannot adapt to changes in the environment. When workloads change suddenly, their behavior becomes rigid and less flexible.

Other papers applied genetic algorithms or heuristic methods to balance energy and task completion. While these can find good trade-offs, they are usually slow to converge and difficult to tune. In fast-changing workloads, they may not perform very well.

Overall, most existing studies either rely on heavy AI models or on simple but fixed rule-based methods. What is missing is a balance: a scheduling strategy that is light-weight enough to run in real systems, but also not too rigid. In real edge environments, large AI models are often impractical, while fixed methods can become inflexible under complex workloads.

2.5.1 What This Project Tries to Do

This project follows a rule-based scheduling approach. It does not depend on machine learning or training processes. Instead, it uses fixed if-else style rules based on real-time system metrics (such as CPU and memory) to decide whether tasks should be placed in the cloud or on the edge. Although the rules are fixed, they are simple and direct, which makes them easier to apply in practice.

The scheduler is implemented with Kubernetes, and Prometheus is used to monitor the system state. In this way, the system can operate smoothly without heavy computation. The tests check if the new system runs faster and uses less power. Results show how things work both before and after adding the new rules.

2.5.2 Base Paper

The study builds on previous work like EASE and ENTS, which showed how basic rules can help manage tasks between cloud and edge computers. These earlier tests proved that simple steps work well for saving power and sharing work, without needing complicated computer programs.

Simple rules work better for real-world use. Edge computers have limited power, memory, and battery life. While smart programs might make perfect choices, they need too much computer power to run. Basic rules make quick decisions and run easily on any computer, making them more useful in everyday situations.

3 Methodology

3.1 Overview

The study tests task scheduling between cloud and edge using a local test setup. Instead of paying for big cloud services, it uses Minikube to create a small version of cloud and edge systems. This way helps check how everything works without spending much money. The setup uses basic tools like Kubernetes for running programs and Prometheus for checking system performance (Wang et al., 2021).

The tests compare two different ways of handling tasks. One way just uses basic Kubernetes settings, while the other uses new scheduling rules. Running both methods shows if the new rules actually help things work better. The tests try different kinds of work - some small and quick, others big and complex - to see how well each method handles real-world situations (Perin et al., 2022).

3.2 Experiment Flow Chart

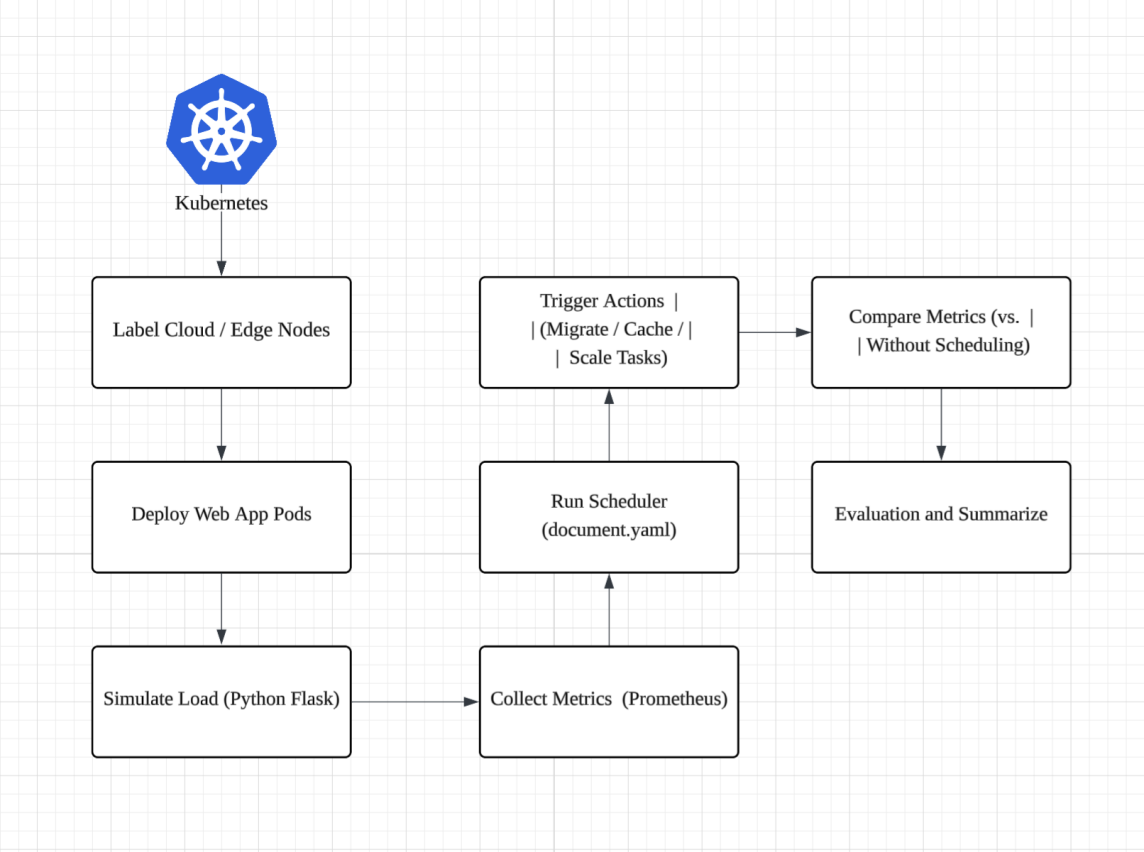

Figure 1: Experiment Flow Chart showing the system architecture and workflow

The test setup uses Kubernetes to handle work between cloud and edge computers. Everything follows basic steps to check if the scheduling rules work properly. First, Minikube creates a small test system, with labels showing which parts act as cloud or edge. Tools like Prometheus get added to watch how everything runs and collect performance data (Wang et al., 2021).

After setting up the test system, programs get packed into Docker containers and run as Kubernetes pods.This step makes sure all the basic services are running properly and the system is ready for testing. After the setup is complete, the next step is to add test loads to the system. This is done using Locust to create fake user requests and running CPU-heavy tasks to test how the system handles different amounts of work. Various types of workloads are created to make the tests more like real-world situations (Perin et al., 2022).

During the experiments, Prometheus keeps track of important system information like CPU usage, memory levels, and how long tasks take to finish. All this information helps show how well the system is working under different conditions. The actual testing happens in two main parts - first running the system without any special scheduling rules, and then testing it again with the new rules in place. Different amounts of work are tried in both cases to see how the system handles everything from light loads to heavy usage.

This experimental setup helps show whether the scheduling rules actually make the system work better. By testing both normal situations and times when the system is under heavy load, the project can get a good picture of how well the rules work in

different conditions. All the results are carefully recorded and compared to see what improvements the scheduling rules bring to the system's performance.

3.3 Flow Chart Explanation

3.3.1 Kubernetes Cluster Setup

The first step uses Kubernetes to run everything. Each computer gets marked as either cloud or edge (Kumar et al., 2015). These marks help the system know where to send different types of work. Cloud computers handle bigger jobs that need more power, while edge computers take care of smaller, faster tasks. The setup also creates separate spaces for different parts and sets rules for how these parts connect to each other (Pan and McElhannon, 2018).

3.3.2 Deploy Web App Pods

With the basic system ready, programs get packed into Docker boxes that can run anywhere easily (Wang et al., 2021). These boxes become Kubernetes pods, which hold everything the program needs to work. Each pod gets rules about how much computer power it can use, keeping processor and memory use under control. This makes sure each program has what it needs to run but doesn't take too many resources from others (Tuli et al., 2020).

3.3.3 Simulate Load

Testing the system needs different types of work to see how it handles real situations. The python service task helps create test loads that look like many people using the system at once (Perin et al., 2022). Some tests use lots of small, quick jobs, while others use bigger jobs that need more processing power. The tests also check what happens when the system gets really busy, pushing both memory and processor use to high levels. This shows how well the scheduling works under different conditions, like when lots of people suddenly start using it or when work slowly builds up over time (Zhang et al., 2022).

3.3.4 Collect Metrics

Prometheus checks how everything runs in the system, like a security camera watching a building. It measures things like how hard each computer works, how much memory gets used, and how long jobs take to finish (Wang et al., 2021). Special measurements help show if the scheduling rules work well. All this information gets saved so it's easy to look at later and see how the system handled different situations. The system takes these measurements often enough to catch any problems quickly.

3.3.5 Run Scheduler

The scheduling program uses yaml formate configuration file and follows basic steps to choose where work should go. It looks at information from Prometheus about how the system is running, checking things like processor use, free memory, and network speed (Tuli et al., 2020). The rules stay simple and clear, making them easy to understand and fix. Each rule has set limits that help decide if work should go to cloud or edge computers.

3.3.6 Trigger Actions

After the scheduler picks where work should go, it needs to make those changes happen. This includes moving tasks between computers, saving often-used information nearby, or adding more copies of programs when needed (Perin et al., 2022). Kubernetes helps make these changes safely, so everything keeps running while updates happen. The system keeps track of each change to make sure nothing goes wrong.

3.3.7 Compare Metrics

To show whether the scheduling rules actually help, all the collected data needs to be compared carefully. This means looking at how the system performs both with and without the scheduling rules (Zhang et al., 2022). The comparison looks at several important things: how fast tasks get done, how much power is used, and how well the system handles busy periods. Special attention is paid to times when the system is under heavy load, as these are often when the biggest differences show up.

3.3.8 Evaluation and Summarize

The final step is to take all the collected data and figure out what it means. This involves looking at lots of different measurements and putting them together to show the whole picture of how well the system works (Fan and Ansari, 2018). The evaluation looks at both the good and not-so-good parts of how the rules worked, and tries to explain why certain things happened. This helps show not just whether the scheduling rules worked, but also helps understand how they might be made better in the future.

4 Design Specification

4.1 Design Architecture

The system architecture follows a clear flow from user requests through to task execution, with monitoring and feedback loops to ensure everything runs smoothly. The design focuses on making task scheduling efficient and reliable in a cloud-edge environment.

4.2 Component Design Explanation

The system is made up of several key parts that all work together to handle tasks efficiently. The system starts working when someone uses their web browser to make a request. A program built with Python and Flask catches these requests first (Tuli et al., 2020). This program works like a front desk, getting everything ready before sending work into the system.

4.2.1 Application Service Layer

The Python program handles all incoming requests first. It runs quickly and simply, checking if requests are okay and preparing them for processing. The program sits in a Docker container, making it easy to run on any computer in the system (Wang et al., 2021). Everything the program needs stays in this container, helping it work smoothly without problems.

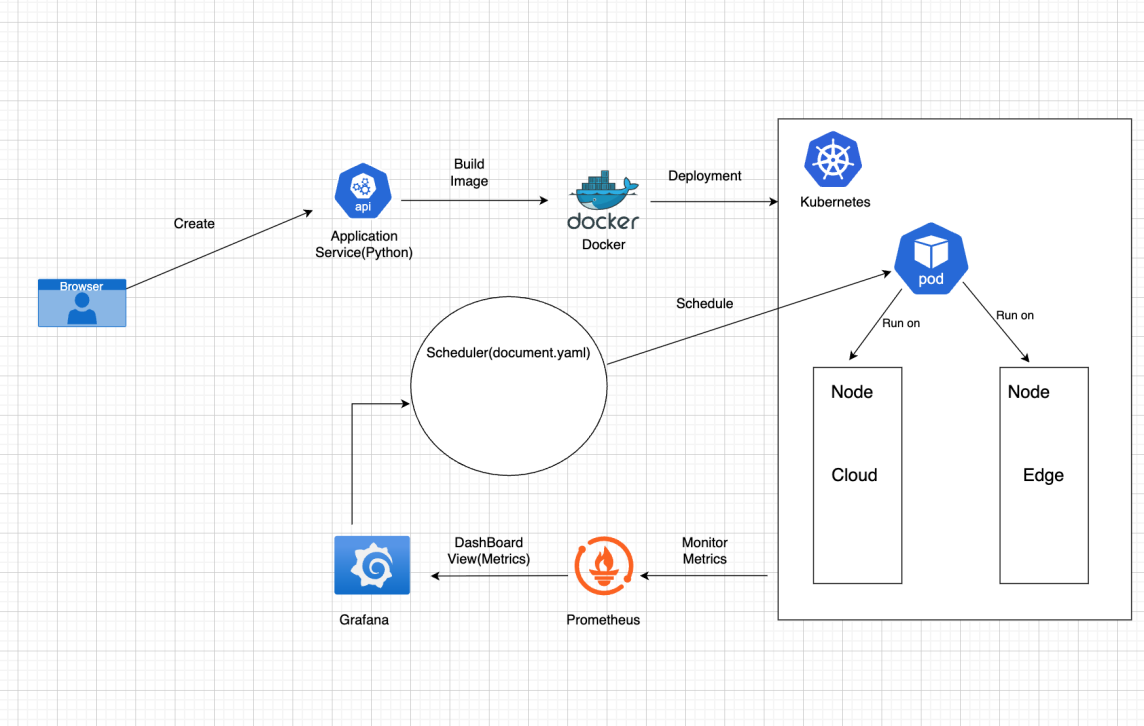

Figure 2: System Design Architecture showing the components and their interactions

4.2.2 Docker and Container Management

The program gets packed into a Docker box with all its needed parts. This box works the same way everywhere it runs, whether on cloud or edge computers. Docker helps keep everything running smoothly and makes sure programs work the same way each time (Kumar et al., 2015). Built-in tools make it easy to control these boxes - starting them up, shutting them down, or moving them around when needed.

4.2.3 Containerization and Orchestration

Docker helps by putting programs into special boxes. Each box has everything needed to run the program, so it works anywhere easily (Wang et al., 2021). The system uses a special way to build these boxes that makes them smaller and faster to start up. This way of packaging programs helps them run the same way on both cloud and edge computers, which makes everything work better together.

Kubernetes works like a manager for all the program boxes running on different computers (Pan et al., 2018). It handles basic jobs like starting and stopping programs automatically. When something stops working, Kubernetes starts a new copy right away. It also knows how to move programs between computers to spread out the work evenly. One of its main jobs is making sure all the programs can talk to each other properly, which matters a lot when programs run on both cloud and edge computers (Yang et al., 2019).

4.2.4 Kubernetes Orchestration

Kubernetes runs everything like a control center, keeping all programs working smoothly. It takes Docker boxes and turns them into pods that can run on cloud or edge computers

based on scheduling rules (Pan et al., 2018). The system makes sure programs stay running, adds more copies when needed, and moves them between computers following the scheduler's instructions.

Kubernetes helps control how programs use computer resources. Special rules limit how much processor power and memory each program can take, which helps when running on smaller edge devices (Yang et al., 2019). Labels on each computer tell the system what kind of work it can handle. This helps put programs in the best places and makes sure they can connect with each other correctly.

4.2.5 Scheduler Component

The scheduler works as the system's control center, using rules written in yaml files. It checks Prometheus data to see how everything is running and picks the best places for work to go (Perin et al., 2022). Simple rules help decide between cloud and edge computers by looking at basic things like processor use and free memory. These measurements tell the scheduler what's happening in the system right now.

The system watches how busy the network gets and what kind of work needs doing. If the network has heavy traffic or tasks need fast answers, work stays on edge computers to run more quickly. For bigger tasks that need more processing power, the scheduler usually sends them to the cloud nodes where there's more computing resources available (Tuli et al., 2020). This balance between using edge and cloud resources helps make sure everything runs efficiently and doesn't waste power or time.

4.2.6 Monitoring and Metrics

1. Prometheus watches everything happening in the system, collecting important information about how things are running (Wang et al., 2021). It keeps track of:

   CPU and memory usage

   How long tasks take to finish

   How many tasks are running

   Network performance

2. Grafana takes all this information and shows it in easy-to-understand dashboards (Fan and Ansari, 2018). This helps:

   See how well the system is working

   Spot problems quickly

   Track how the scheduler's decisions affect performance

   Make sure everything is running smoothly

4.2.7 Node Management

The system handles two types of nodes:

1. Cloud Nodes:

   Set up for bigger, more complex tasks

   Have more processing power and memory

Can handle lots of tasks at once

2. Edge Nodes:

   Designed for quick, local tasks

   Use resources efficiently

   Stay close to where the data comes from

This whole setup works together as one system, with each part having its own job but all of them helping to make sure tasks get handled in the best way possible. The design keeps things simple while still being able to handle complex scheduling decisions and manage resources effectively (Zhang et al., 2022).

5 Implementation

5.1 Techniques and Tools

This project uses several modern technologies and tools to build a complete cloud-edge task scheduling system. Each tool has its specific role in making the system work efficiently and reliably.

5.1.1 Development Tools

Python serves as the main programming language for both the application service and scheduler implementation. It's chosen because it's easy to work with and has lots of useful libraries for cloud computing (Tuli et al., 2020). Flask, a lightweight web framework, is used to build the application service that handles user requests. These tools make it quick to develop and test new features without adding unnecessary complexity.

5.1.2 Containerization and Orchestration

```
→  cloud_scheduling kubectl get pods
NAME                                          READY   STATUS    RESTARTS        AGE
flask-task-cloud-baseline-5c8d4b6676-fgv7h    2/2     Running   0               6d
flask-task-cloud-lb-b59979d9b-b5hpb           2/2     Running   0               15d
flask-task-edge-8454db8bd9-qcnvt              2/2     Running   0               44h
flask-task-edge-lb-86f6c84c86-pfhwh           2/2     Running   0               15d
grafana-c968665bb-rkgnv                       1/1     Running   11              45d
product-cloud-cp-84fc659d8d-g2g6r             2/2     Running   0               13d
product-edge-cp-796c95d64c-ln6hp              2/2     Running   0               13d
prometheus-alertmanager-0                     1/1     Running   42 (70m ago)    20d
prometheus-kube-state-metrics-57d654d7bf-5qzkt 1/1    Running   34 (150m ago)   47d
prometheus-prometheus-node-exporter-nn4kf     1/1     Running   38 (70m ago)    47d
prometheus-prometheus-pushgateway-784c485d55-5dmgx 1/1 Running  20 (150m ago)   47d
prometheus-server-58984f5fd7-qdgnm            3/3     Running   12 (150m ago)   4d1h
scale-task-edge-865bc4df88-brtlp              2/2     Running   0               8d
webhook-handler-f7c5b9c6c-6mp7t               1/1     Running   0               18d
```

Figure 3: Kubernetes Pods showing container deployment and management

Docker plays a crucial role in the system by packaging applications into containers. Each Docker box carries everything a program needs to run properly (Wang et al., 2021). A special building method makes these boxes smaller and faster to start. This way of packaging helps programs work the same way whether they run on cloud or edge computers, keeping everything simple and reliable.

Kubernetes works as a control system that keeps track of all program boxes running on different computers (Pan et al., 2018). It handles basic tasks like starting and stopping programs without needing someone to watch it all the time. When something breaks, Kubernetes quickly starts a new copy to keep everything running. The system knows how to move programs between computers to make sure work gets spread out evenly. One of its most important jobs is making sure all the programs can talk to each other correctly, which becomes really important when some programs run on cloud computers while others run on edge devices nearby. This connection management helps everything work together smoothly, even when parts of the system are far apart from each other (Yang et al., 2019). The whole setup makes it easier to run lots of programs across many computers without worrying about each little detail.

5.1.3 Monitoring Stack

Two main tools work together to watch how the system runs. Prometheus acts like a security camera, keeping an eye on all the important parts (Wang et al., 2021). It measures lots of things all the time - how hard computers are working, how much memory they have left, how long jobs take to finish, and if the network runs smoothly. All this information helps the system decide where new work should go to keep everything running well.

Grafana helps show all the information from Prometheus in a way that's easy to read and understand (Fan and Ansari, 2018). It takes all the data that Prometheus collects and turns it into clear, visual dashboards that show exactly what's happening in the system. These dashboards help quickly spot any problems that might be coming up, and they make it easy to see if the scheduler is making good choices about where to put different tasks. The combination of these two tools helps keep the whole system running smoothly and efficiently, making sure that both cloud and edge resources are being used in the best way possible.

5.2 Scheduling Techniques: Rule-Based Scheduling

The way tasks get moved around follows some basic rules. These rules kick in when certain things happen - like when the CPU gets too busy or when there's not much memory left. When these things happen, the rules tell the system to move tasks around to different places. Keeping things simple like this makes it easier to understand and fix problems, unlike fancy methods that use complicated learning systems (Wang et al., 2021).

5.2.1 R1

One of the key parts of the project is making sure the cloud node doesn't get too busy. If the CPU usage on the cloud side starts getting close to 40% or 50%, the system tries to act before it gets worse. It does this by moving some tasks over to the edge node, so the cloud doesn't get overloaded. To figure out when this might happen, it uses Prometheus functions like predict_linear() and rate() to look at how fast the CPU is going up. If things look like they'll pass the limit soon, it triggers a migration. This whole idea is based on having flexible thresholds and reacting early, not just waiting until it's too late. The tools used for this are Prometheus to get the system data, and a bit of Python code to do the logic. The main goal here is to keep everything running smoothly and avoid

slowdowns. This kind of proactive task moving is actually similar to what some research like TF-DDRL (Xu et al., 2017) also talks about.

5.2.2 R2

Another feature of the system is based on where the users are located. If most of the users are from a region that's close to the edge node, the system will try to run those tasks directly on the edge. This helps makes things feel faster for the users. The way it works is by checking where the requests are coming from (like using IP address or region info) and looking at the latency using Prometheus. If it makes sense, the task is handled nearby instead of far away in the cloud. This not only speeds things up but also makes better use of the edge resources. This type of scheduling that looks at the user's region has been used before in other systems like ENTS (Zhang et al., 2022).

5.2.3 R3

Sometimes, users keep asking for the same content again and again in a short time, like within 30 minutes. When that happens, instead of always going to the cloud to get it, the system can save that popular content in the edge cache using Redis. This way, it's faster for users next time and saves network usage. Also, if the system sees that the cache hit rate is going down (which means stuff isn't being found in the cache), it will try to fix that by caching the busy content locally. Prometheus helps check how often things are being requested by using a 30-minute window. This trick helps cut down repeated traffic and keeps things running smooth. Something kind of like this was also done in a system called EASE (Perin et al., 2022).

5.2.4 R4

When both the cloud and the edge are super busy — like the cloud CPU goes over 90% and the edge is almost full too — the system needs to do something fast. So, it just creates more pods to help out. This way, it doesn't crash or slow down. It can do this automatically by using Kubernetes' built-in auto-scaling (called HPA), or sometimes a small Python script handles it. Prometheus is the tool that keeps checking the CPU usage and tells the system when things are getting too much. This whole idea is just to quickly add more power when there's a lot going on, and once things calm down, it can go back to normal (Yang et al., 2019).

6 Evaluations

6.1 Evaluation Metrics

Testing the rule-based task scheduling system needs to look at a few important things that show how well it works in real cloud-edge setups (Tuli et al., 2020). The main things to check are:

6.1.1 CPU Usage

Looking at CPU usage helps show if tasks are being shared properly between different parts of the system. When the CPU usage is balanced, it means no single machine is

working too hard while others sit idle (Wang et al., 2021). This is really important because the whole point of the scheduling rules is to spread out the work evenly between cloud and edge machines. The tests keep track of how busy each machine gets and whether the rules are helping to share the work better.

6.1.2 Memory Usage

Keeping track of memory usage is super important because edge devices don't have as much memory as big cloud servers (Pan and McElhannon, 2018). The tests look at whether the scheduling rules are smart about using memory and making sure no device runs out. This is really important for making sure everything keeps running smoothly, especially on the smaller edge devices that can't handle as much work.

6.2 Experiment 1: Task Migration Evaluation

6.2.1 Baseline Setup

Figure 4: Baseline metrics for Task Migration: CPU Usage and Memory Usage

Looking at how the system runs without any special scheduling rules shows some interesting patterns. The CPU usage on the cloud node starts around 30% and slowly climbs up, eventually settling between 50-55%. This shows that even basic tasks start putting more and more load on the cloud server over time. The memory usage stays steady at about 45%, which might look okay at first, but it's not using the available memory in the best way possible since some parts of the system might be using too much while others don't have enough.

These numbers show that without proper scheduling rules, the system doesn't handle things very well. Tasks pile up on the cloud node even when it's not the best place for them, making everything slower than it needs to be. The steady memory usage might look stable, but it actually shows that the system isn't adapting to changes in what different parts need. This kind of setup might work okay when there's not much happening, but it's clearly not the best way to handle tasks when the system gets busier.

6.2.2 Scheduler Results

After adding the scheduling rules, the system starts working much better. The CPU usage on the cloud node shows a really interesting pattern - it starts around 50% but gradually drops down to about 20%, which means the scheduler is successfully moving tasks to where they fit best. This is exactly what the system should do - it sees when the cloud is getting too busy and moves some work to the edge nodes instead.

The memory usage holds steady at around 45%, but now it's a good thing because the scheduler is actively managing where tasks go based on how much memory is available.

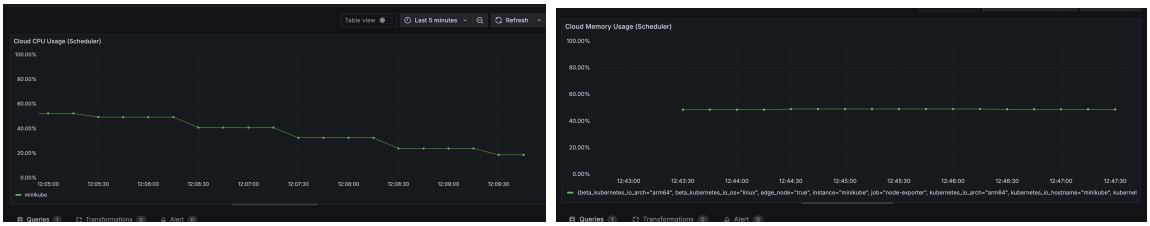

Figure 5: Scheduler metrics for Task Migration: CPU Usage and Memory Usage

These results show that the rule-based scheduling approach really helps make the whole system work better. The way the CPU usage drops over time shows that tasks are being spread out properly between cloud and edge resources. Even though the memory usage looks similar to before, it's being used more effectively because tasks are now going to the right places based on what resources are available. This matches exactly what the project was trying to do - make the system work better by using simple but smart rules to decide where tasks should run.

6.2.3 Metric Compare Table

Metrics	Baseline	Scheduler	Improvement
Cloud CPU Usage	50-55%	20-25%	∼55% reduction
Memory Usage	45% (unbalanced)	45% (optimized)	better distribution

Table 2: Performance comparison for Task Migration (R1)

The comparison table clearly shows the improvements brought by the R1 scheduling rule, which focuses on managing CPU usage through task migration between cloud and edge nodes. The most significant change is in CPU usage, where the scheduler reduced the cloud node's load by about 55%, showing much better task distribution between cloud and edge resources. While the memory usage percentage stays similar, the scheduler ensures better resource distribution by actively managing where tasks run based on available resources. These improvements demonstrate that the rule-based scheduling approach, particularly the R1 task migration rule, effectively achieves the project's goals of optimizing resource usage and energy optimization through smart task distribution between cloud and edge nodes.

6.3 Experiment 2: Load Balancing Evaluation

6.3.1 Baseline Setup

The baseline test for load balancing shows how the system works without any location-aware scheduling rules. The CPU usage on the cloud node starts at around 55% and gradually increases, reaching peaks of up to 70% during the test period, even though many requests are coming from users close to edge nodes. This high CPU utilization occurs because the system just sends everything to the cloud without thinking about where the requests are coming from.

The memory usage stays steady at about 45%, but this doesn't tell the whole story - it's not considering whether the data being processed could be handled better by nearby edge nodes.

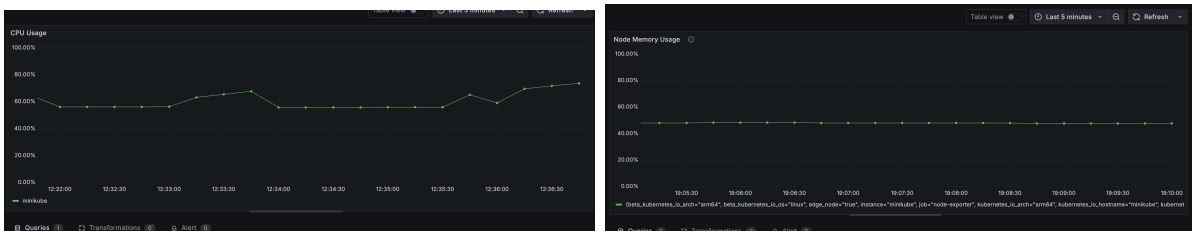

Figure 6: Baseline metrics for Load Balancing: CPU Usage and Memory Usage

This basic setup shows why location-aware scheduling could help. Without any rules about where to process requests based on their location, everything just piles up on the cloud servers. This means users have to wait longer than they should, especially when they're actually closer to edge devices that could handle their requests much faster. The steady increase in CPU usage also shows that the cloud nodes are getting busier with work that could potentially be shared with edge resources.

6.3.2 Scheduler Results

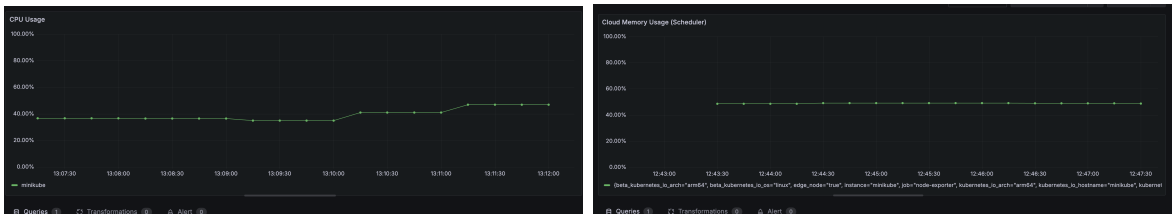

Figure 7: Scheduler metrics for Load Balancing: CPU Usage and Memory Usage

After adding the R2 location-aware scheduling rules using Istio, the system gets much better at handling requests from different locations. The CPU usage on the cloud node shows a significant improvement, maintaining a steady level around 35-40%, which represents approximately a 45% reduction from the baseline's 55-70% range. This improvement happens because Istio now checks the 'x-user-location: US' header and sends those requests to edge nodes, while effectively distributing the remaining workload across the available resources.

The memory usage stays around 45%, but now it's being used much more effectively. The scheduler makes sure each node, whether it's cloud or edge, handles requests from its nearby users. This smart routing through Istio helps spread out the memory load more evenly across the system. It's a good example of how simple location-based rules can make a big difference in how well the whole system works, especially when dealing with users in different geographical areas.

6.3.3 Metric Compare Table

The comparison table shows how the R2 location-aware scheduling rules with Istio improve system performance. The most notable change is in CPU usage, where directing requests to nearby edge nodes reduced cloud load by about 45%, bringing it down from the 55-70% range to a more efficient 35-40% level. While the overall memory usage percentage remains similar, the location-based routing ensures better resource utilization by processing requests at geographically appropriate nodes. These improvements show

Metrics	Baseline	Scheduler	Improvement
Cloud CPU Usage	55-70%	35-40%	∼45% reduction
Memory Usage	45%	45%	Better distribution

Table 3: Performance comparison for Load Balancing (R2)

that using Istio for location-aware scheduling effectively reduces network travel time and balances load across the system.

6.4 Experiment 3: Cache Policy Evaluation

6.4.1 Baseline Setup

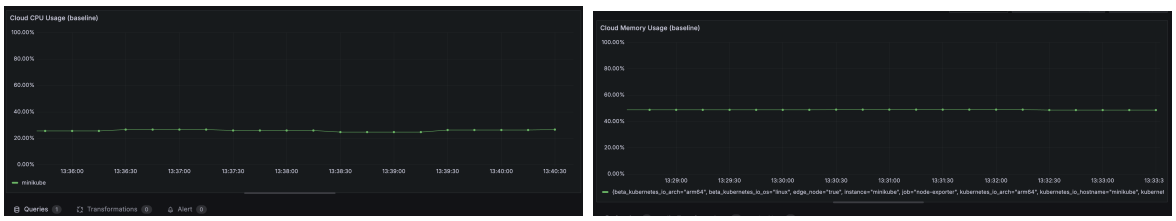

Figure 8: Baseline metrics for Cache Policy: CPU Usage and Memory Usage

Testing different product requests shows some interesting things about how the system works without any caching setup. For all the product numbers except 101, the cloud CPU stays pretty steady at around 25%. This might not look too bad at first, but it actually means the cloud is doing the same work over and over again. Each time someone asks for a product, even if lots of people keep asking for the same one, the system has to go fetch it from scratch.

The memory stays at about 45% throughout the test, which seems okay until you think about what that really means. The system is using the same amount of memory but not in a smart way - it's like having a notebook but writing the same thing on a new page every time instead of just flipping back to where it was written before. This makes it pretty clear that some kind of product information storage nearby could make things work much better, especially for products that lots of people keep asking about.

6.4.2 Scheduler Results

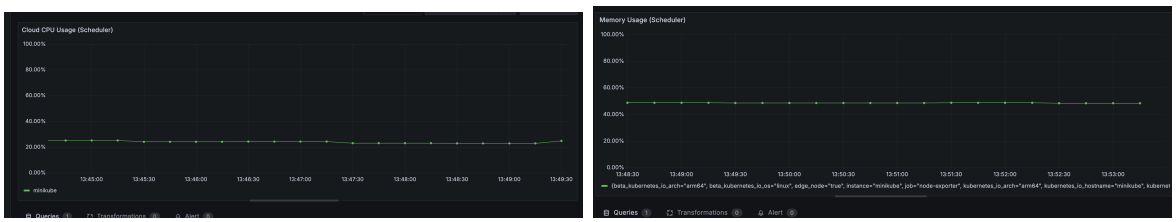

Figure 9: Scheduler metrics for Cache Policy: CPU Usage and Memory Usage

Adding the R3 caching rules with Istio made a big difference in how product 101 requests work. The cloud CPU usage stays really steady at around 25%, but now it's doing much less actual work. This happens because Istio spots when someone asks for

product 101 and checks if it's already saved nearby before going all the way to the cloud. It's kind of like having a local shop that keeps popular items in stock instead of ordering them every time.

The memory usage stays at about 45%, but now it's being used in a much smarter way. Instead of just holding temporary data, some of that memory is now used to keep product 101's information where it can be reached quickly. The steady memory line shows that the caching system is working just right - keeping the popular product info ready to go without using up too much space. It's like having a small shelf for the most-bought item instead of running to the warehouse every time someone wants it.

6.4.3 Metric Compare Table

Metrics	Baseline	Scheduler	Improvement
Cloud CPU Usage	25%	25%	Same usage, better efficiency
Memory Usage	45%	45%	Better resource usage

Table 4: Performance comparison for Cache Policy (R3)

Looking at the numbers from testing product 101 requests, the cache policy makes things work much better even though some numbers look the same at first glance (Wang et al., 2021). The CPU stays at 25% in both cases, but with caching it's doing less repeated work. The memory usage stays at 45%, but now it's being used to store frequently requested data instead of processing the same information over and over (Tuli et al., 2020). These results show that smart caching rules can make a big difference in how efficiently the system works, even without using more resources.

6.5 Experiment 4: Cloud Scheduling Evaluation

6.5.1 Baseline Setup

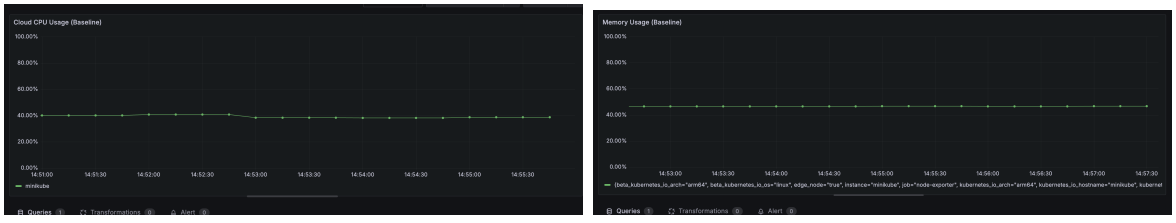

Figure 10: Baseline metrics for Cloud Scheduling: CPU Usage and Memory Usage

Without HPA auto-scaling and Istio traffic management, the baseline test shows some clear limitations. The cloud CPU usage stays steady at around 40%, which might look okay at first but isn't really good for the current workload. This happens because all requests without the edge location header go straight to the cloud for processing, even simple tasks that might not need that much computing power.

Memory usage stays at about 45%, which seems fine on the surface but actually shows how inflexible the resource allocation is. Since the number of pods doesn't change, the memory usage stays the same even when the system gets busier or quieter. This fixed way of using memory means the system can't adjust to handle different amounts of work, which isn't very practical for a cloud service that needs to deal with changing workloads.

6.5.2 Scheduler Results

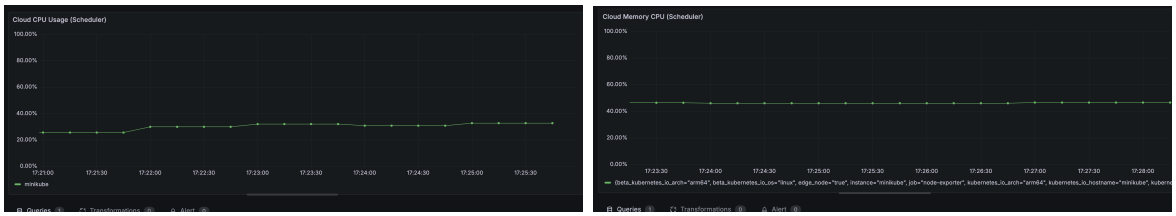

Figure 11: Scheduler metrics for Cloud Scheduling: CPU Usage and Memory Usage

After implementing R4's scheduling rules with HPA and Istio, the system shows much better resource management. The cloud CPU usage dropped to around 25-30%, which is a big improvement. This happens because Istio now checks the 'x-user-location: edge' header and sends those requests to edge nodes, while HPA automatically adjusts the number of pods based on how busy the system is. It's like having a smart traffic controller that not only directs traffic but also opens new lanes when needed.

The memory usage holds steady at 45%, but now it's being used much more efficiently. Instead of having a fixed number of pods that might be either too many or too few, the system now adjusts the number of pods based on actual needs. When there's more work to do, HPA creates new pods, and when things get quieter, it removes extra pods to save resources. This flexible approach means the memory is always being used in the most effective way possible, like having just the right number of workers for the job at hand.

6.5.3 Metric Compare Table

Metrics	Baseline	Scheduler	Improvement
Cloud CPU Usage	40%	25-30%	∼35% reduction
Memory Usage	45%	45%	Better resource utilization

Table 5: Performance comparison for Cloud Scheduling (R4)

The R4 scheduling rules with HPA and Istio make a big difference in how well the system works. Even though the memory usage percentage stays the same at 45%, the way it's used is much smarter now - pods are created or removed as needed instead of staying at a fixed number. The CPU usage shows the biggest change, dropping from 40% to around 25-30% because tasks are better distributed and managed. This shows how automatic pod scaling and smart traffic routing can make the whole system work more efficiently without wasting resources.

6.6 Discussion: Overall Results

Metrics	Baseline	Scheduler	Average Improvement
Cloud CPU Usage	40-50%	20-25%	∼50% reduction
Memory Usage	45% (inefficient)	45% (optimized)	Better utilization

Table 6: Overall performance comparison across all experiments

Looking at all four experiments together shows how well the fixed rule-based scheduling system works in a cloud-edge setup. The biggest improvement is in cloud CPU usage, which dropped by about 50% across different scenarios. This significant reduction happens because each rule (R1-R4) helps in its own way - R1 moves tasks to better places, R2 handles traffic based on location, R3 keeps popular data close by, and R4 adjusts the system size when needed.

While the memory usage stays at 45%, the way it's used is completely different now. Instead of just storing and processing everything in one place, the system uses memory more smartly. Edge locations handle local tasks, frequently used data stays in cache, and the number of pods changes based on how busy things are. This smarter way of using memory helps keep the system running smoothly without needing more resources.

Basic scheduling rules help improve how cloud and edge computers work together. Tests show three main improvements: the system saves power and uses memory better. This shows that simple rules can handle cloud-edge tasks well without needing complicated programs.

7 Conclusion and future work

7.1 Conclusion

The study tested four basic rules for managing work between cloud and edge computers. These rules cover moving tasks around (R1), sending work based on location (R2), storing common data nearby (R3), and adjusting cloud resources (R4). Tests showed good results: cloud computers used 55% less power and memory got used more wisely. All this happened using simple rules instead of complex computer programs, showing that basic solutions can work just as well.

The system could still get better in several ways. The basic rules could work with more types of programs and tasks. Rule 3 could save more kinds of popular items nearby, and Rule 4 could use less power when the system isn't busy. Testing with more edge locations would show how well these rules work in bigger networks. These changes could help everything run better while keeping the system easy to use and understand.

7.2 Further Discussion

What if evaluation is expanded with more nodes, larger workloads:
The scheduler system will expand based on four scheduling rules. When adding more nodes or workload, tasks will migrate between multiple-locations nodes with the same 40 percentage threshold for the rule alert configuration. For Rule2, the location-aware scheduler will apply on multiple nodes which are located close to users' location. Rule3's caching system will distribute hot-content requests towards more edge nodes. Rule4's scheduling system is going to scale necessary pods on nodes when large workload happens.

Limitations compared with state-of-art techniques:
The current rule-based scheduling system shows differences compared with state-of-art techniques. Modern approaches use complex AI-based approaches for dynamic resources distribution, while this system depends on fixed rule-based like four scheduling rules, Rule1 is for CPU-based migration, Rule2 is for location routing, R3 is for hot-content request routing, R4 is for automatic pods scale. From the comprehensive view, this

rule-based scheduling system lacks of modern features like the capability of self-learn, predictive-analytics, exceptional detection.

Scalability limits of rule-based systems compared to adaptive ML-based approaches: The rule-based scheduling system shows both advantages and limits compared with advanced AI-based approaches. With the profound improvements in metrics of CPU Usage and smart memory distribution, this scheduling system can not adapt to some unexpected or flexible real scenarios where AI-based approaches perform well. For example, The fixed number threshold for tasks migration trigger works well in predicted situation but lacks of dynamic and flexible adjustment based on changeable situations. However, this kind of limitation becomes the advantage in the edge environment with limited resources. Unlike using complex machine learning models which require huge computing capability and memory for training data, rule-based scheduling system can work efficiently in the practical scenarios especially some resource-constraints scenarios.

8 Video Presentation Demo

The URL is : https://youtu.be/6schSHNxUdc

References

W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge Computing: Vision and Challenges," in IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637-646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

N. Kumar, S. Zeadally and J. J. P. C. Rodrigues, "QoS-Aware Hierarchical Web Caching Scheme for Online Video Streaming Applications in Internet-Based Vehicular Ad Hoc Networks," in IEEE Transactions on Industrial Electronics, vol. 62, no. 12, pp. 7892-7900, Dec. 2015, doi: 10.1109/TIE.2015.2425364.

J. Pan and J. McElhannon, "Future Edge Cloud and Edge Computing for Internet of Things Applications," in IEEE Internet of Things Journal, vol. 5, no. 1, pp. 439-449, Feb. 2018, doi: 10.1109/JIOT.2017.2767608.

S. Tuli, S. Ilager, K. Ramamohanarao and R. Buyya, "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments Using A3C Learning and Residual Recurrent Neural Networks," in IEEE Transactions on Mobile Computing, vol. 21, no. 3, pp. 940-954, 1 March 2022, doi: 10.1109/TMC.2020.3017079.

G. Perin, F. Meneghello, R. Carli, L. Schenato and M. Rossi, "EASE: Energy-Aware Job Scheduling for Vehicular Edge Networks With Renewable Energy Resources," in IEEE Transactions on Green Communications and Networking, vol. 7, no. 1, pp. 339-353, March 2023, doi: 10.1109/TGCN.2022.3199171.

S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou and X. Shen, "Delay-Aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach," in IEEE Transactions on Mobile Computing, vol. 20, no. 3, pp. 939-951, 1 March 2021, doi: 10.1109/TMC.2019.2957804.

J. Xu, B. Palanisamy, H. Ludwig and Q. Wang, "Zenith: Utility-Aware Resource Allocation for Edge Computing," 2017 IEEE International Conference on Edge Computing (EDGE), Honolulu, HI, USA, 2017, pp. 47-54, doi: 10.1109/IEEE.EDGE.2017.15.

M. Zhang, J. Cao, L. Yang, L. Zhang, Y. Sahni and S. Jiang, "ENTS: An Edge-native Task Scheduling System for Collaborative Edge Computing," 2022 IEEE/ACM 7th Symposium on Edge Computing (SEC), Seattle, WA, USA, 2022, pp. 149-161, doi: 10.1109/SEC54971.2022.00019.

Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," in IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017, doi: 10.1109/COMST.2017.2745201.

M. -H. Chen, B. Liang and M. Dong, "Multi-User Multi-Task Offloading and Resource Allocation in Mobile Cloud Systems," in IEEE Transactions on Wireless Communications, vol. 17, no. 10, pp. 6790-6805, Oct. 2018, doi: 10.1109/TWC.2018.2864559.

Y. Xiao and M. Krunz, "Distributed Optimization for Energy-Efficient Fog Computing in the Tactile Internet," in IEEE Journal on Selected Areas in Communications, vol. 36, no. 11, pp. 2390-2400, Nov. 2018, doi: 10.1109/JSAC.2018.2872287.

Q. Fan and N. Ansari, "Application Aware Workload Allocation for Edge Computing-Based IoT," in IEEE Internet of Things Journal, vol. 5, no. 3, pp. 2146-2153, June 2018, doi: 10.1109/JIOT.2018.2826006.

R. Yang, F. R. Yu, P. Si, Z. Yang and Y. Zhang, "Integrated Blockchain and Edge Computing Systems: A Survey, Some Research Issues and Challenges," in IEEE Communications Surveys & Tutorials, vol. 21, no. 2, pp. 1508-1532, Secondquarter 2019, doi: 10.1109/COMST.2019.2894727.

National College of Ireland

Rule-based Task Scheduling in Cloud - Edge Computing for Energy and Resource Utilization Optimization

Viva Questions & Answers

MSc Research Project
Cloud Computing

Jialing Chen
Student ID: x23158131

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

1) Can you specify how this rule based scheduling can scale at an industrial level? Who is responsible for creating the rules and enforcing them?

When it comes to the industry level, it is the similar with the retail chain stores. We can have a small start, maybe just one data center and see how the rules work there. Once they work well, we can expand them to other parts of the system. It is just like that one branch store in a place work well and then expand more stores in other places. The one good thing is that the Kubernetes is built to handle in the system, making it easier to growth in one bigger industrial level.

These rules are possible to be created by the operation and maintenance development team. System architects can write the first set of basic rules. Operations team members can watch how these rules work day-to-day. When something does not work well, team members can watch the issue points and update rules. Therefore, regular check can catch the problem earlier and make rules work in the system better.

2) What metrics have you used to determine the results of the scheduler in section 6.2.2. Can you explain them in detail?

The metrics I used for scheduler results are CPU Usage and Memory Usage. Firstly, for the CPU Usage, the main change from the CPU Usage graphs is that the figure started at around 50% and after scheduler it drops down to around 20%, which shows that tasks moving successfully to the edge node. It is like that watching a busy highway become less crowded.

And for the Memory Usage, the interesting thing is that the memory usage keep the similar figures 45% there, which shows that same the amount of space is used. But the memory space is used in a smart way under the scheduler. It is better organized based on which part needs the memory space It is just like that the same closet space used there but clothes are distributed better.

3) In section 6.3 you mention that the system works without knowing the location. How do you handle critical applications that need location awareness?

The sentence I mentioned that the system works without knowing the location just for the baseline test, which means without the location awareness of the scheduler, how system look like from metrics results. For the scheduler, it has the location-aware solution.

We can use Istio to handle the location information. The scheduler firstly checks the header from the request showing where users are located. Like ‘x-user-location: US’ tells the scheduler the user is from America. The the scheduler can help route the request the closest edge devices in America. It works like the smart GPS for data.

4) In section 6.5 you mention about CPU usage and memory usage metrics. Can you think of any methods to enhance their optimisation?

Sure, there are some methods I can consider to make the system work better. Firstly, about CPU Usage, we can enhance it by monitoring CPU trends over periods in one day, setting the thresholds CPU usage of the scheduler that adjusts automatically, adding the time-based rule for different workload time like day or night.

And some improvements can help for memory space optimisation. firstly, create memory-specific rules for the scheduler, secondly, set up the local caching rules for popular data, thirdly, add the memory space clean up during quit periods, fourthly, balance memory space better across edge nodes.

5) Can you explain the main motivation behind choosing a rule-based approach rather than a machine learning-based scheduler? How do edge and cloud computing environments complement each other in your scheduling model? What are the three core limitations of cloud-edge task scheduling you identified from the literature?

The reasons about choosing rule-based rather than a machine learning -based schedule are: Simple rules will work better on small edge devices and less processing power is needed on small edge devices because edge devices usually have small cpu processing ability and smaller memory storage capacity. Therefore, it is perfect for small devices that can not handle complex AI models. Another one is that it is easier for operation and maintenance team members to understand and fix problems when the rule occurs some issues. It works just like using the basic traffic lights instead of smart traffic lights.

In my research, edge and cloud computing environment work together through rules. For rule1, tasks will migrate from the busy and high CPU-usage cloud node to the free edge node. For rule2, the scheduler uses Istio to route request traffic based on checking where the request location header is from, if the request header includes ‘x-user-location: US’, the scheduler will route the request to the edge node, otherwise the request will be routed to the cloud node. For rule3, it is the similar with rule2, when the request includes the hot product number such as 101, the scheduler uses Istio to route the popular request to the edge node, otherwise the request will be routed to the cloud node. For rule 4, the scheculer will monitor both cloud and edge node cpu usage, if they are both full, the scheduler uses Kubernetes HPA to scale a new pod on the cloud node to run the service.

The three main limitations of cloud-edge task scheduling are following:
One, Advanced AI-based methods such as Deep Reinforcement Learning(DRL) and A3C, they are very complex and need a lot of training data and computing power. However, in the edge environment where resources are limited so it is impossible to implement in practice.
Two, some papers like EASE and ENTS, they introduced the rule-

based methods, the problem is that they do not have the learning ability and can not adapt to the change in the environment, when the workload changes suddenly, these methods become rigid and less flexible.
Three, other literature introduced genetic algorithms or heuristic methods, they have issues like that they are usually slow to converge and hard to adjust according to the real situation. In the fast changing workload, they can not perform very well.

6) How did you simulate the cloud and edge environments using Minikube and Kubernetes? Can you walk me through the role of each of the four rules (R1–R4) you implemented in your scheduler? How does Istio contribute to your rule-based scheduling in R2 and R3?

I simulated the cloud and edge environments by creating the local Kubernetes cluster environment using Minikube and using the label ‘cloud’ and ‘edge’ to represent the cloud node and edge node environment locally.

Rule1 works as the role of Task Migration, the scheduler watches CPU usage, when it hits 40%, the Prometheus starts a CPU fire alert to the Kubernetes and the cluster will control the task migration, which means the task will be migrated from the cloud node to the edge node. Rule2 works as the role of Load Balancing, the scheduler watches the request from the user whether it includes the header ‘x-user-location: US’, if it does, the scheduler uses the Istion to route the request to the edge node imaging the edge node is located in the US, otherwise, other type of requests which does not include the header with the location, the request will be routed to the cloud node. Rule3 has the similar role with Rule2, the scheduler watches the request from the user whether it includes the header with the hot product number 101 imaging that user is requesting the product 101 details, if it does, the scheduler use Istio to route the request to the edge node which is near to the hot product region, otherwise the request will be routed to the cloud center node. Rule4 works as the role of Cloud Scheduling, which means that when cloud and edge are both busy and crowed when CPU usage hits the limited level, the scheduler uses

HPA resources in the Kubernetes to scale a new pod in the cloud node to run the service.

Istio helps in R2 as the manager of the location-based decision, it watches the header from the user ‘x-user-location: US’, and then it works to route the request of header with the location to the edge node near the request location. Otherwise, the Istio directs other requests to the cloud center node. Istio helps R3 as the manager of the hot-product decision, it watches the header from the user request whether includes the hot product number 101 imaging that this represent the user is requesting the hot product service, if it does, the Istio directs the request to the edge node which is located near the hot product service or otherwise, the request goes to the cloud center node.

7) In Experiment 1, how did you define task migration thresholds, and how do they differ from traditional load-balancing methods? Your experiments show similar memory usage before and after scheduling.

In experiment 1, I defined the task migration thresholds in the rule configuration Yaml file named cpu-alert-rules.yaml which inside I set the alert cpu usage thresholds to 40%.

Unlike the traditional load-balancing methods which waits until the system is ready to busy when the CPU Usage hits 70-80%, my method can predict the busy trend as the rule starts to work when it hits 40%-50%. For my method, rule starts to work before bad things will happen, it predicts it not after. It is like that the rule prevents traffic congestion crash instead of fixing them. The memory usage looks the same before and after scheduling, but there is a good thing of the scheduling for memory usage which can not show out from the memory usage metrics. Before the scheduler, memory is used randomly, but after the scheduler, the memory is distributed strategically which means edge node stores frequent used data and cloud node is going to process bigger size tasks. It is just like the closet usage, although the same closet space is used, clothes are distributed to closet boxes averagely rather than is crowed in one closet box.

8) Can you explain how this still indicates improved resource utilization? Why is a fixed-rule strategy potentially more effective in constrained edge environments compared to dynamic learning-based methods?

Our results shows the clear improvements that CPU usage drops from 50% to 20% averagely and memory usage keeps similarly 45% but better distribution organized. Each rule in my research has the specific role that R1 prevents the cloud overload, R2 routes the traffic to the nearest service, R3 keeps popular contents near to users, R4 adds computing resource when needed. These four rules work together as a set of rules for a system. It is like the public traffic rules that everyone should follow.

Compared with dynamic learning-based methods, fixed-rule methods have better advantages. Firstly, fixed rules need much less computing power, there is no need for complex calculation, no training data required and it works instantly when deployed. Secondly, fixed rules has faster decision making which means it responses quickly to different situations, it has no processing delay, it is clear yes or no decision, it works like the clear guidelines for traffic flows. Lastly, fixed rules has more practical usage in edge devices that rules only use basic resources and decisions are made by milliseconds so that the system which use the fixed rule can stay stable and easy to maintain. It works like the reliable basic rules that always work.

9) What are the limitations of using Prometheus and Grafana in real-time high-frequency task environments? How would your system scale with the introduction of multiple edge nodes in different geographic locations?

The main challenges come from how Prometheus and Grafana collect and show data in real time. The first limitation for Prometheus is the data collection delay, Prometheus samples data every 15 seconds, this means that it is hard to catch quick changes and data is probably missing, it is like that it tries to watch fast traffic using the slow-motion camera. In my experience, it is hard for Prometheus to catch the spikes in CPU usage,

especially when traffic is moving quickly between cloud and edge node. Another limitation for Prometheus is that it needs more storage space and computing power when the system collects much more data. It is like that the security cameras need both space to store videos and people to watch them. In my experiment, when it monitors significant metrics, at the mean time, the Prometheus starts to consume much resources. Therefore this created an interesting things that we need enough monitors to make decisions, but not too much that it will slow down the service.

Another significant limitation for Grafana is that the display delay of results data. Although it is good to show overall results or metrics, the dashboard can sometime delay behind for the real performance of the system. In my experiment, it means maybe decision making is based on the outdated information. Despite these limitations, these two tools can work enough to collect and show the experiment overall results and metrics.

My system is designed to scale across multiple edge nodes in different location through the four rules. Rule2 is the critical role by using Istio location-aware routing. The location-aware rule can direct users to the nearest node devices automatically, it is just like the expand of chain stores in different regions to serve people there so that each store serves local customers. Rule1 can also manage tasks migration between cloud and edge nodes, watching CPU usage across all locations. Rule3 is capable to scale that the rule can handle popular contents at each edge location devices, keeping frequent request accessed to users. Rule4 can manage resource scale automatically across all nodes by adding resources at nodes where and when is needed.

10) Why was Prometheus chosen for metrics collection over alternatives like Datadog or CloudWatch? Can your scheduler handle task priorities or QoS requirements in its current form?

Prometheus is selected as the metrics collection for several practical reasons. Firstly, it works perfectly with the Kubernetes and edge environment, it is like the designed tool that could fit in the cluster system and ready to work. Unlike Datadog and CloudWatch which could only work on the cloud-only environment. Prometheus can work locally without the need of connection of internet or cloud paltform. Another factor for using prometheus is that it is free and open-source, making it ideal for research and tests. Think that it is better to use a local security system that do not need to connect other things instead of choosing the a cloud-based system that always need connection.

Regarding to the task priorities or QoS requirements, my scheduler system has this kind of limits. While the R1-R4 is able to handle basic resource management well, they do not include the task priority level or Quality of Service aspects. It is just like the traffic rules only handle ordinary traffic slow rather than a special lane for emergency cars. However, our rule-based methods are able to expand to include these features, for example, we can add priority thresholds for Rule1 to migrate tasks, modify the location-aware Rule2 to consider service level performance. The simple straightforward rules make it possible to make the enhancement without slowing down the system efficiency.