National College of Ireland

Custom Kubernetes Scheduler on Raspberry Pi for IoT Applications, Optimizing Node Selection Based on Energy Efficiency.

MSc Research Project
Cloud Computing

Kotagiri Venkata Durga Abhinav
Student ID: 23344997

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Kotagiri Venkata Durga Abhinav
Student ID:	23344997
Programme:	Cloud Computing
Year:	2024-2025
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	15/09/2025
Project Title:	Custom Kubernetes Scheduler on Raspberry Pi for IoT Applications, Optimizing Node Selection Based on Energy Efficiency.
Word Count:	2597
Page Count:	21

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:	Kotagiri Venkata Durga Abhinav
Date:	15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

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

Custom Kubernetes Scheduler on Raspberry Pi for IoT Applications, Optimizing Node Selection Based on Energy Efficiency.

Kotagiri Venkata Durga Abhinav
23344997

Abstract

This study proposes the deployment of a Kubernetes cluster using Raspberry Pi devices with a custom Kubernetes scheduler specifically designed for edge environments, with the help of Kubernetes scheduler plugins. The primary objective of the study is to select the nodes by incorporating real-time bandwidth and energy metrics to deploy the workloads, which helps in improving the performance and reducing resource wastage. With today's growing adoption of the Internet of Things, fog and edge paradigms, there is a demand for efficient and low-power edge computing infrastructure. Raspberry Pi devices, even though they are resource-constrained, offer a cost-effective solution for deploying containerized applications or workloads in an edge environment. However, the default Kubernetes scheduler considers only the CPU and memory utilization metrics, which are not enough for the better optimization of these devices to run IoT applications. In this paper, the methodology involves collecting the custom node metrics using open-source tools such as Prometheus, for monitoring the node's energy usage, and Iperf for the node's bandwidth measurement, these metrics are used and integrated with a plugin-based scheduler framework to score and rank nodes more intelligently than the default Kubernetes scheduler. Through the experiment, by comparing the default scheduler with the new scheduler, the new scheduler shows measurable improvements in bandwidth cost efficiency and energy-aware workload distribution in the Raspberry Pis. Results are visualized with the help of Grafana dashboards, and these findings highlight the effectiveness of the multi-metric scheduling approach compared to the traditional approach in optimizing Kubernetes for fog and edge computing scenarios.

Keywords: Kubernetes, Raspberry Pi, Bandwidth Optimization, Energy Efficiency, Prometheus, Grafana, iPerf, Container Orchestration.

1 Introduction

1.1 Research Background

With the increasing adoption of the Internet of Things (IoT), there is a need for deploying lightweight, scalable, and secure applications in edge environments. Traditional cloud computing infrastructure has limitations in serving latency-sensitive and bandwidth-sensitive IoT applications due to network distance and communication factors. Due to this, container orchestration tools like Kubernetes have emerged as the best option

in managing the distributed workloads efficiently in both cloud and edge environments. Kubernetes is widely used in cloud environments compared to edge environments, but its default scheduler, which mainly considers CPU and memory usage, is not optimized for edge use cases where energy constraints and network bandwidth are important factors.

Raspberry Pi devices, which are cost-effective microcomputers and known for their low power consumption and compatibility with containerized applications Kayal (2020), offer great compatibility for edge computing. Prior research has demonstrated that Kubernetes clusters can be deployed on Raspberry Pi devices. However, challenges remain in optimizing the scheduling decisions while deploying the workloads. This paper addresses that challenge by exploring a multi-metric custom scheduler using the Kubernetes custom scheduler framework, specifically designed for edge environments.

1.2 Motivation

The Kubernetes default scheduler does not take crucial metrics like bandwidth and energy efficiency, which are critical parameters while selecting the nodes for deploying the workloads. Ignoring these factors often results in inefficient workload distribution, power wastage, and higher application response time. By including real-time bandwidth and energy metrics, this paper aims to bridge a significant gap in the present edge orchestration strategies.

The practical implementations of this study propose a cost-effective alternative to the other high-end edge orchestration solutions by enhancing the Kubernetes performance on low-cost hardware. This aligns with the sustainability goals in computing and encourages the broader adoption of edge computing for smart homes, smart cities, and real-time IoT applications.

1.3 Problem Statement

The standard Kubernetes scheduler has some constraints, as it mainly selects a node based on the CPU and memory usage metrics Kayal (2020). These drawbacks become a problem in the edge environments, where factors such as network performance and power efficiency are as crucial as computational capacity. To address this gap, this study introduces a custom Kubernetes scheduler developed using the Kubernetes Scheduling Framework. The proposed solution complements the default scheduler by including additional node-level metrics, such as bandwidth and energy consumption, into the decision-making process. Node bandwidth is evaluated using iperf, while Prometheus collects real-time cluster metrics.

1.4 Research Question

Can a custom Kubernetes scheduler, which is incorporated with energy efficiency metrics, outperform the default scheduler in terms of communication overhead and workload distribution efficiency in an edge environment?

1.5 Proposed Solution

The study introduces the development of a custom Kubernetes scheduler for Raspberry Pi clusters, designed specifically for edge computing environments. The key problem addressed is that the default Kubernetes scheduler only considers CPU and memory usage,

which is insufficient for edge scenarios where energy efficiency and network bandwidth are critical factors. To overcome this limitation, the proposed solution involves: Deploying a Kubernetes cluster on Raspberry Pi devices, forming a low-cost, lightweight edge computing infrastructure, and integrating real-time bandwidth and energy consumption metrics into the scheduler's decision-making process using: Prometheus for energy monitoring and iPerf for bandwidth measurement.

1.6 Objectives

In this paper, the following objectives have been established: investigating the current state of the Kubernetes scheduling within the fog and edge computing domains, designing a custom scheduler with the help of the Kubernetes scheduler plugins framework that integrates the bandwidth and energy metrics, these metrics are monitored with the help of Prometheus and evaluate the new schedulers performance against the default scheduler using controlled deployment on a Raspberry Pi cluster.

- Investigate the current limitations of Kubernetes scheduling in fog and edge computing environments.
- Design and implement a custom Kubernetes scheduler using the Kubernetes Scheduling Framework and plugin architecture.
- Integrate bandwidth availability and energy consumption as primary node-level metrics for scheduling decisions.
- Use Prometheus for real-time energy monitoring and iPerf for bandwidth measurement across Raspberry Pi nodes.
- Install a lightweight Kubernetes cluster using K3s on Raspberry Pi devices.
- Visualization of performance and system metrics using Grafana.

1.7 Limitations

This study uses a hardware setup with Raspberry Pi devices and does not involve real-world production environments. While Prometheus and iperf offer robust monitoring capabilities, they introduce a slight performance overhead, and finally, the scope is limited to Kubernetes deployments using the k3s lightweight distribution.

1.8 Paper Structure

This paper is organized into six key sections. Section 2 provides a comprehensive review of related work. This section highlights the current situation in Kubernetes scheduling in fog and edge environments and identifies research gaps. Section 3 outlines the methodology adopted for implementing the custom scheduler, including project setup and tools used. Section 4 presents the design specifications of the proposed system, describing the architecture and hardware configuration. Section 5 details the implementation process. Section 6 evaluates the performance of the custom scheduler compared to the default Kubernetes scheduler with experiments and using the available metrics. Finally, Section 7 concludes the study by summarizing the findings and proposing some directions for future work.

2 Related Work

2.1 Kubernetes in Fog and Edge Environments

In the paper Gizinski and Cao (2022) demonstrated a practical experiment of deploying edge computing workloads using the Raspberry Pi devices was demonstrated, and also in this study shows that local processing at the edge can significantly reduce the data transmission latency, which is very important in time-sensitive IoT applications. By using Raspberry Pis as edge servers, their research demonstrated local and parallel processing capabilities and illustrated that edge computing can outperform traditional cloud setups under specific workloads.

In this paper, Kayal (2020) addresses the growing demand for leveraging Kubernetes, a widely used container orchestration tool, in the context of fog and edge computing. Fog computing has been in demand for the last decade as it decentralizes the cloud resources, pushing computation and data storage closer to the data source in order to reduce latency and network overhead. While Kubernetes has proved a very good and smart tool in cloud environments, the paper identifies numerous limitations for the edge setup.

This paper begins by defining a fog computing model composed of distributed nodes such as Raspberry Pi devices, which act as fog nodes capable of hosting containerized applications. The practical implementation of this paper consists of deploying a containerized IoT application across the Raspberry Pi cluster. Two deployment strategies are examined: placing the entire application on a single fog node versus distributing the microservices across multiple fog nodes. The experimental results reveal that both approaches are feasible but expose latency trade-offs in distributed deployments due to the network communication overhead.

This paper identified the architectural limitations of the Kubernetes default scheduling policies and the specific needs of fog environments. The paper describes that the default Kubernetes scheduler is unaware of constraints such as bandwidth and node energy metrics and other factors that significantly impact performance in edge environments. While Kubernetes can technically be run on low-power devices using distributions like K3s, MicroK8s, or KubeEdge, these solutions address orchestration overhead but not the scheduling logic.

Deploying a Kubernetes cluster on a Raspberry Pi is explored by the Füstös et al. (2022), where this paper presents a wide approach to setting up and operating a light-weight, scalable Kubernetes cluster using Raspberry Pi devices. Using K3s, a light-weight Kubernetes distribution specifically optimized for the ARM architectures, the authors build a four-node cluster using an Ansible playbook. Ansible is an open-source configuration management tool. They showcased two proof-of-concept applications to illustrate real-world workload deployment. The system architecture includes Rancher for cluster management, and monitoring is done by using Prometheus and Grafana dashboards for the graphical representation. The infrastructure uses static IPs and a private network, enhanced with SSH security measures and GitHub-based user access control. This contribution is particularly useful for prototyping scheduling algorithms in edge environments, where hardware constraints, power efficiency, on low-cost devices are primary concerns. Though the paper doesn't propose a new scheduling algorithm, its deployment insights are crucial for experimenting with edge-native container orchestration models.

This paper Haja et al. (2019) addresses some of the challenges that we are facing in implementing Kubernetes in the edge environment, where low latency and delay sensitiv-

ity play an important role. While Kubernetes has emerged as the go-to tool for container orchestration in cloud-based applications, its default design and its default scheduler lack optimization for the geographically distributed and resource-constrained devices, and also, the edge devices are prone to failure. In this paper author proposes an extension to the default Kubernetes that introduces delay-aware scheduling and reliability mechanisms that are needed for edge computing. The innovation that they have done is a custom Kubernetes scheduler that takes and adds the real-time latency measurements between the edge nodes. They did this by deploying the lightweight pods called ping pods across nodes. The newly implemented system monitors the round-trip delays, constructing a delay matrix that informs scheduling decisions. Nodes are now labeled and ranked based on these measurements, which enables the scheduler to make placement choices that satisfy the application's needs.

2.2 Scheduling in Kubernetes For Fog and Edge Environments

Carrión (2022) provides a comprehensive survey and taxonomy of the Kubernetes scheduling strategies. The study classifies container orchestration into infrastructure, application, and performance metrics and identifies a research gap in scheduling for fog and edge environments. In the context of fog and edge computing, this study is very important as it highlights the need for the adaptation of Kubernetes' default scheduling mechanism to meet the unique requirements of decentralized environments.

The research begins with a foundational overview of Kubernetes' architecture. It defines how Kubernetes functions using a centralized control plane, including the scheduler, which is responsible for the pod placement decisions. These decisions, by default, have a two-phase approach: filtering out unsuitable nodes based on the node resources and a scoring to rank the remaining nodes using criteria like CPU, memory, and affinity rules.

While the traditional model is efficient for cloud-native environments, it fails to address the issue in fog and edge computing. The default scheduler and the general-purpose design limit the Kubernetes performance in environments where nodes have different architectures, which are subject to constraints like energy consumption and network connections.

This paper reviews the multiple scheduling extensions and custom strategies proposed in recent years. Some key techniques include: energy-aware Scheduling, which incorporates power consumption metrics into scheduling decisions. For example, KEIDS (Kubernetes-based Energy and Interference Driven Scheduler) introduces energy and interference constraints to support sustainable industrial IoT operations.

The KEIDS scheduler Kaur et al. (2020) addresses two highly important challenges in Industrial Internet of Things (IIoT) Kubernetes deployments, which are energy efficiency and container interference. Existing Kubernetes frameworks struggle with the co-scheduling of multiple resource-intensive workloads and applications across the edge and cloud infrastructure while ensuring minimal power consumption and high performance. To bridge this gap, the authors proposed a multi-objective optimization problem modeled using Integer Linear Programming to minimize the carbon emissions, energy use, and workloads in containerized applications.

KEIDS operates in two phases: scheduling and synchronization. In the scheduling phase, containerized applications are assigned to nodes using a mathematical model that incorporates green energy sources, for example, solar and wind energy, resource limits,

job deadlines, and interference metrics. The next phase, Synchronization, ensures that the system's actual state aligns with its desired state using Kubernetes' etcd metadata service. The scheduler accounts for three container workload categories: CPU-intensive, I/O-intensive, and network-intensive, each influencing the scheduling and interference considerations.

The study Senjab et al. (2023) presents a categorization of Kubernetes scheduling literature into specific use cases, and those are generic scheduling, which is a default strategy with just basic optimization, and multi-objective optimization, which has algorithms balancing resource utilization and latency. The paper also demonstrates artificial intelligence-based scheduling using machine learning for predictive decisions and autoscaling-aware scheduling using dynamic resource allocation. The paper critically evaluated these approaches and underlined the lack of work specifically for the fog and edge environments, despite the growing adoption in these contexts.

2.3 Latency-Aware and Cost-Optimized Scheduling in Kubernetes

This paper Eidenbenz et al. (2020) describes a fog layer architecture that implements a latency-aware scheduling into Kubernetes for industrial automation use cases. Unlike generic cloud workloads, industrial IoT applications often require very low latencies and heterogeneous infrastructure. The proposed system introduces an algorithm to minimize latency between interconnected components in a distributed fog network.

In this paper, applications are modeled as graphs where the edge nodes represent communication demand and the nodes correspond to compute units with resource constraints. Here, the goal is to place these devices, foglets, onto fog nodes such that the overall network latency is minimized. The key challenge addressed here is Kubernetes's inherent lack of awareness about the underlying network topology and latency. Some Kubernetes native mechanisms, like affinity rules, are used to address this limitation.

Li et al. (2023), a Kubernetes native scheduling extension designed to optimize microservice deployments by using the spot and on-demand instances in cloud environments. Recognizing that not all microservices have equal criticality, some microservices are divided into three priority levels and strategically placed low-priority services on cost-effective spot instances, while ensuring high availability for high-priority services through resilient on-demand nodes.

This architecture includes the three major components, and those are: a scheduler, a daemonset to monitor spot instance evictions, and cronjobs to orchestrate node pool scaling based on pre-defined service rules. The affected services are rapidly rescheduled using over-provisioned secondary nodes. This hybrid approach achieves both cost reduction and minimal downtime.

Cost optimization scheduling in Kubernetes is evolving in the field of container orchestration. The Arunan et al. (2023) solves the growing concerns of managing the cloud infrastructure costs while maintaining the high availability for the microservices-based applications that are deployed on Kubernetes. This paper recognized that not all microservices will have the same computing demand, and the authors proposed KubeEconomy, a novel system that intelligently leverages a combination of spot and on-demand virtual machines to reduce operational expenses without compromising the system's reliability.

2.4 Custom Scheduling in Kubernetes

Effective scheduling using the custom scheduler has been explored Chandra (2023), and Kubernetes has become a dominant orchestration framework for containerized applications due to its robust capabilities for workload management, scalability, autoscaling, and automation. According to some surveys, adoption has grown steadily, with major cloud providers such as AWS, GCP, and Azure offering managed Kubernetes services. The paper focuses on the default scheduling approach of Kubernetes, emphasizing the new scheduler's use of a two-step process involving filtering and scoring to determine the optimal placement of workloads on nodes. During filtering, strict requirements are applied to exclude the unsuitable nodes based on some factors such as resource availability. The node with the highest score is selected to host the workloads. Although this approach provides reasonable baseline performance, it has limitations, particularly in its static nature and lack of awareness of real-time resource utilization. Many research efforts have explored the development of custom schedulers to address these gaps.

Recognizing these limitations, the paper identifies the need for a custom scheduler that incorporates real-time memory usage metrics when making pod placement decisions. This paper proposed a work to fill the gap by developing a custom Kubernetes scheduler that utilizes Prometheus-collected real-time memory metrics to optimize pod placement.

2.5 IoT Devices Security with Kubernetes and Raspberry Pi Cluster

Donca et al. (2024) presents a comprehensive security framework for IoT devices using a Kubernetes-managed Raspberry Pi cluster and advanced security tools. The paper addresses the growing demand for secure and scalable IoT deployments, mainly in security vulnerabilities of the distributed IoT infrastructure.

The authors designed and implemented an architecture that integrates BME680 environmental sensors with a Raspberry Pi cluster orchestrated by Kubernetes. Giving priority to security risks associated with IoT, mainly concerning unauthorized access and hard-coded credentials. The proposed system adopts a multi-layered security approach to safeguard both devices and data.

Two security technologies are used to solve these limitations. First, Hashicorp Vault is utilized for dynamic secrets management, replacing hard-coded credentials, which significantly reduces the risk of credential compromise. Second, OpenID Connect (OIDC) is integrated to provide strong, standardized authentication across the system.

2.6 Scheduling and Containerization Applications in Resource-Constrained Environments

There is related work done on containerization applications, especially on resource-constrained devices. Kang et al. (2021) study compared the container orchestration tools such as Kubernetes and Docker Swarm on Raspberry Pi-based IoT gateways, focusing on smart home and industrial automation use cases. This study evaluated the performance metrics such as container creation time, CPU usage, and memory consumption. The study offers practical deployment guidelines for maintaining service availability in fog and edge environments using Kubernetes.

The study Schnider and Hutter (2024) proposes an energy-aware scheduling improvement for Kubernetes that considers renewable energy availability on cluster nodes. The authors incorporate battery and solar generation data into scheduling decisions to favor energy-efficient node usage. This kind of scheduling contributes to sustainable computing by reducing grid dependency and maximizing renewable energy utilization. Canova (2024) work aims to bridge the gap between traditional scheduling and formal optimization scheduling. The author developed a scheduler extension using tools like Google OR-Tools and Z3 solver to resolve the unscheduled pods. This study selects ideal pod and node placements using the resource usage and affinity rules, and by this, it is clear that it is feasible to integrate smart optimization into Kubernetes.

Table 1: Summary of Research Proposals

Author(s) and Year	**Research Proposal**	**Algorithm**	**Dataset**	**Tool**	**Limitations**
Arunan et al. (2023)	Cost optimization for microservices in Kubernetes	KubeEconomy	Simulated workloads	Kubernetes, Spot VMs	May suffer downtime for low-priority services
Donca et al. (2024)	Secure IoT data collection using RPi and Kubernetes	RBAC, OIDC and Vault integration	BME680 sensor data	Kubernetes, Vault, OIDC	Limited to Raspberry Pi and AWS setup
Schnider and Hutter (2024)	Energy-aware scheduling for green computing	Custom scheduler with energy metrics	Solar energy logs	Kubernetes	Limited scalability beyond 16 nodes.
Canova (2024)	Suboptimal default K8s scheduling under resource limits	OrTools optimization plugin	Synthetic workloads	Kubernetes, OrTools, Z3	only tested on small clusters
Chandra (2023)	Improve memory utilization in K8s	Custom memory-aware scheduler	Live node metrics	Kubernetes and Prometheus	Only tested on EC2, no CPU tuning
Sugunakumar et al. (2023)	Reduce microservices cloud costs	KubeEconomy with spot instance rebalance	K8s microservices	Kubernetes	Relies on excess capacity, 2-min downtime risk
Stan et al. (2024)	Secure IoT monitoring using Raspberry Pi cluster	Multi-layer security stack with Vault and OIDC	BME680 environmental sensor data	Kubernetes, Vault, OIDC, Docker, AWS	Not scheduling-focused

Sharpening K8s (2019)	Latency-aware edge K8s	Delay-aware heuristic and pod placeholder	Synthetic latency traces	Modified Kubernetes	Limited failure tolerance validation
Kuljeet et al. (2020)	Energy and interference aware IIoT scheduler	ILP-based KEIDS	Data centers	Kubernetes, ILP	Not scalable beyond medium cluster
Raphael et al. (2020)	Latency-aware orchestration for fog	Heuristic fog scheduler	Industrial control traces	Kubernetes	No vendor-specific tuning; synthetic testing
Fustos et al. (2022)	Edge orchestration	Ansible and K3s	Two sample apps	Prometheus and Grafana	Limited to 4 RPi and small test size
Kang et al. (2021)	Container orchestration	Empirical comparison	IDS workloads	Docker, Kubernetes	Swarm lacks orchestration features and high latency
Elbaz et al. (2023)	Cost-efficient scheduling in K8s clusters	Beetle Antennae Search optimization algorithm	Simulated workloads	Kubernetes	No hardware validation; high convergence time for large clusters
Islam et al. (2023)	Survey of K8s scheduling strategies and taxonomies	Comparative analysis	Literature Papers	kubernetes	No experimental validation; focuses only on theoretical comparison

2.7 Summary

Kayal (2020) demonstrated Kubernetes for fog and edge computing environments using Raspberry Pi; this paper also identified the limitations in latency-sensitive deployments and proposed architectural enhancements, but the study is limited to the centralized scheduling model, where the default scheduler considers memory and CPU usage only and does not solve the real-world scalability problems. Work done by Gizinski and Cao (2022) also uses Raspberry Pi as the edge servers to reduce the data transmission time through local and parallel processing. Although this project is a unique approach in edge processing environments, it lacks scalability and does not address different types of workloads. Carrión (2022) provided a comprehensive taxonomy and critical analysis of Kubernetes scheduling strategies, gaps in edge-specific solutions, but the work is entirely theoretical and lacks real-world implementation. This project addresses these limitations by developing a real-time metrics-aware scheduler with network metric integration using real workloads that is optimized for low-power energy-sensitive clusters.

Base Papers:

1. Gizinski, T. and Cao, X., 2022, January. Design, implementation and performance of an edge computing prototype using raspberry pis. In 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC) (pp. 0592-0601). IEEE.
2. Kayal, P., 2020, June. Kubernetes in fog computing: Feasibility demonstration, limitations and improvement scope. In 2020 IEEE 6th World Forum on Internet of Things (WF-IoT) (pp. 1-6). IEEE.

3 Methodology

3.1 Proposed Solution

The project introduces a novel approach to developing a custom Kubernetes scheduler designed for Raspberry Pi clusters. The cluster nodes are configured so that they operate as lightweight and low-cost edge nodes, which are capable of running containerized microservices or applications. The main objective of this paper is to improve the efficiency of the pod placement decisions in Kubernetes by adding additional node-level metrics, such as bandwidth and energy, which are, by default are not considered by the default scheduler.

This solution is needed as there are limitations to the default Kubernetes scheduler, which primarily deploys pods on nodes based on compute and memory resources. while it is a good option for the high-resource cloud environments, only considering these metrics is insufficient for resource-constrained edge computing environments where network conditions and energy consumption play a crucial role. Edge devices, such as Raspberry Pis and other embedded compute devices, operate on a low power budget and are subject to varying network conditions, making context-aware scheduling is required for efficient and sustainable system operations.

To address these challenges, a custom scheduler is implemented using the Kubernetes scheduling plugin framework. This framework is used for scheduling decisions without modifying the core architecture of Kubernetes. The custom scheduler supplements the existing scheduling decisions by evaluating each node within the cluster based on the three key metrics, which are: energy consumption, bandwidth availability, and standard resource availability, like CPU and RAM.

Raspberry Pi devices, chosen for their affordability and compatibility with containerized workloads these devices are formed as an edge-based cluster. All the Raspberry Pis have been installed with Ubuntu 24.04 as the operating system, which has been installed on the micro SD card using the Raspberry Pi Imager, and they also have the necessary tools installed, such as iPerf, which is a network performance measurement tool. This architecture has many advantages, from reducing data transmission to centralized cloud servers, it also reduces latency, conserves bandwidth, and improves the performance of IoT applications.

3.2 Project Setup

The proposed custom Kubernetes scheduler requires a well-equipped and well-planned setup for both hardware and software components to ensure that the setup is reliable and reproducible. This section outlines the key activities involved in preparing the experimental setup, including the installation of the Kubernetes cluster and integration with monitoring tools. To deploy the containerized applications at the edge environment,

a Kubernetes cluster is deployed on Raspberry Pis using K3s, a lightweight Kubernetes distribution that is well optimized for resource-constrained devices such as the Raspberry Pis. K3s is used in this setup because it gives many advantages in edge computing environments, those are simplified installation and compatibility on devices that have minimal system requirements, making it well-suited for ARM-based architectures, as Raspberry Pis have an ARM-based CPU.

The cluster follows the master and worker architecture similar to the regular Kubernetes, where one Raspberry Pi acts as the master node and the other as the worker node. Before installing K3s, all the Raspberry Pi devices are installed with the latest compatible operating system, which is Ubuntu 24.04 for ARM architecture. K3s is installed using the commands provided by Rancher Labs from the official documentation to minimize manual configuration errors. All the worker nodes will join the cluster using the master node's unique cluster token.

3.3 Custom Scheduler Development

A custom Kubernetes scheduler is developed using the Kubernetes scheduling framework, specifically utilizing the scheduler extender or plugin architecture. The scheduler retrieves real-time metrics from Prometheus and iPerf. A scheduling score is computed for each node based on Available bandwidth, Real-time energy consumption, and Standard resource availability, such as CPU and RAM The scheduler overrides the default node selection logic in Kubernetes, ranking nodes higher if they exhibit better throughput and lower energy consumption. The custom scheduler is developed as an extension to the Kubernetes scheduling process by leveraging the Scheduler Extender and the Plugin architecture provided by the Kubernetes Scheduling Framework.

3.4 Evaluation Process

The effectiveness of the custom scheduler is assessed through a comparative experimental setup. Both the default and custom schedulers are tested under a similar kind of workload. All the Performance metrics, including pod scheduling time, bandwidth, and energy efficiency of the nodes, are recorded and analyzed. Results are visualized through Grafana, confirming improvements in resource optimization and reduction in communication overhead by the custom scheduler.

3.5 Data Description and Ethical Considerations

This project uses an application and a publicly available data to make sure that the project can be reproducible and the project is done with fair transparency and ethical compliance. Two types of the datasets used are the bandwidth and energy metrics and workload data. This project follows the very important and crucial ethical considerations at all stages of the project, such as data ownership and consent. Security of edge devices and transparency. From these practices, the project maintains ethical integrity while developing the energy and bandwidth-aware scheduling in Kubernetes for edge environments.

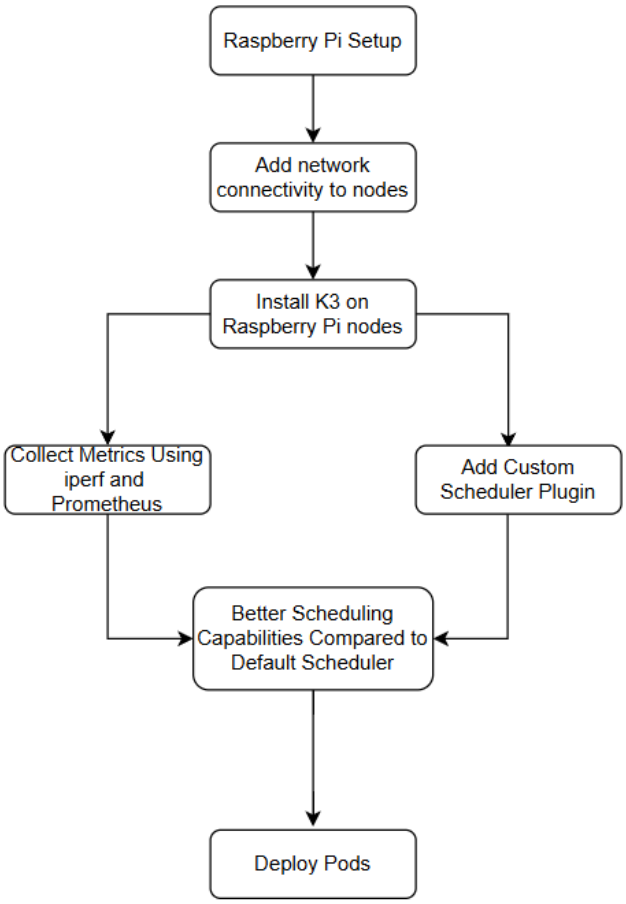

Figure 1: Methodology Flow Chart

4 Design Specification

The architectural design of the proposed project is based on a lightweight Kubernetes, K3s, deployed on Raspberry Pi devices. These devices are deployed as master and worker devices. This design aims to optimize scheduling of workloads in fog and edge environments by adding real-time bandwidth and energy metrics into the Kubernetes scheduling process. The Kubernetes distribution selected for implementation is K3s, a low-memory-consumption and production-compatible Kubernetes distribution developed by Rancher Labs. For fog and edge environments, K3s is the best option due to its low binary size, low memory, and simple installation. All these reasons make K3s the ideal choice as a container orchestration tool to install on Raspberry Pis. In the proposed setup, the cluster consists of three Raspberry Pi devices, one as the master device and the other two as the worker devices. The master node is also called the control plane, will have the custom scheduler plugin, and worker nodes will host the containerized workloads.

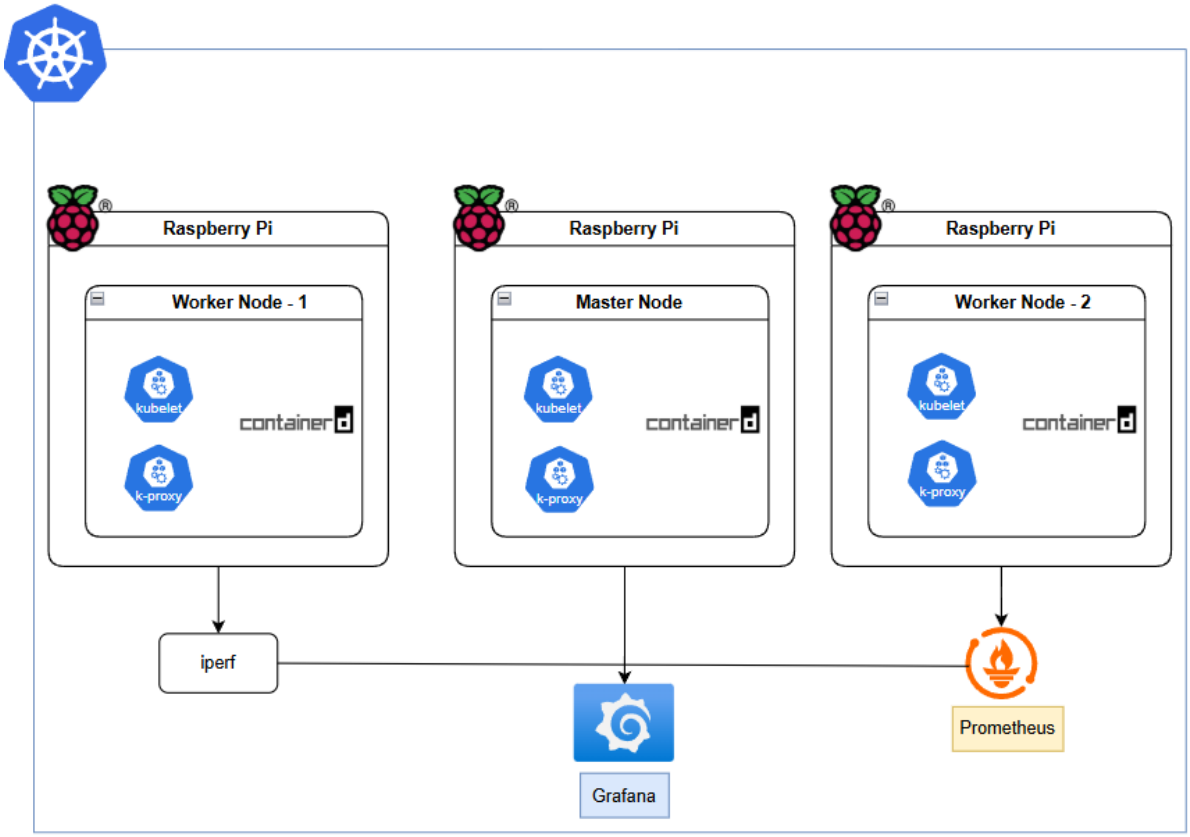

Figure 2: Bandwidth, and Energy Aware Scheduling

4.1 Overview

4.2 Devices configuration

All the Raspberry Pi nodes are configured with essential Kubernetes components, including kubelet, kube-proxy, and containerd. The work of the kubelet is to run the containers that are defined in the pod specification. Kube-proxy handles the networking within the cluster, and containerd serves as the container runtime. on the master node, a custom scheduling algorithm is developed by using the Kubernetes Scheduler Framework. This scheduler works alongside the default scheduler. The main task of this scheduler is to improve the workload placements, mainly pods, by not only CPU and RAM utilization, which are the default metrics used by Kubernetes, but also the real-time bandwidth and energy consumption of the devices. These additional metrics are very important in edge environments where devices are very power sensitive and have limited networking.

Two tools are used to collect the required metrics: bandwidth and energy consumption. The first is iperf, a widely used open-source tool to measure network bandwidth. This tool is installed on the devices to generate bandwidth data between nodes. The other tool is Prometheus Node Exporter, an open-source event monitoring and alerting tool that collects metrics such as CPU utilization, CPU temperature, and system load. All these metrics are gathered by a central Prometheus server, which pulls data at fixed intervals. Grafana, an open-source analytics and visualization platform, is used along with Prometheus to provide real-time visualization of both bandwidth and energy metrics.

4.3 Working Principles of Scheduler

The custom scheduler has two main principles to do those are filtering and scoring the nodes. During the filtering stage, nodes that do not meet the minimum resource requirements or have a low bandwidth are excluded from consideration to host the pods. This process ensures that only the eligible nodes are passed to the next stage, scoring stage. In the scoring stage, each node is assigned a score based on all the cumulative values of the available metrics. Nodes with high bandwidth and lower energy consumption are ranked higher. With the help of the scoring mechanism, the pods are hosted on the nodes that have a high score, which have sufficient resources and are also optimal in terms of communication speed and energy usage.

This design introduces a novel context-aware scheduling approach in Kubernetes. By integrating real-time metrics into scheduling decisions, it improves workload placement in edge environments. This design uses K3s and open-source monitoring tools; all these things make the solution lightweight and replicable for other edge computing scenarios. Additionally, using Raspberry Pi devices provides a cost-effective platform and makes the setup accessible to the research community.

5 Implementation

5.1 Implementation Overview

The final stage of the project is about implementing and deploying the custom Kubernetes scheduler on a lightweight edge computing infrastructure. In this project, the primary infrastructure is Raspberry Pi devices. This scheduler is designed specifically to improve workload placements, such as pods, by adding real-time metrics such as bandwidth and energy efficiency, which are not considered by the default Kubernetes scheduler. By adding all these metrics, the system will improve performance, reduce power consumption, and optimize network usage. Some of these are challenging factors in the fog and edge environments.

5.2 Tools and Technologies

The Kubernetes cluster was deployed on Raspberry Pi using K3s, which is a lightweight distribution of Kubernetes that is best suited for devices such as Raspberry Pi and Beagleboard. The cluster follows the master and worker architecture, with one Raspberry Pi working as master and the other devices as workers. A careful selection of tools and technologies is made to ensure they are compatible with Raspberry Pi devices. The first goal was to develop an ecosystem where real-time system metrics could be reliably collected for pod scheduling decisions in Kubernetes. These tools are efficient and easily integrable with each other.

K3s is a lightweight Kubernetes distribution specifically optimized for ARM-based single-board computers like Raspberry Pi. It reduces the resource overhead compared to a full Kubernetes installation by removing unnecessary features compared to the traditional cloud provider installation. So, this is one of the best Kubernetes distributions that is suitable for IoT and edge environments where performance and energy efficiency are very critical.

To achieve intelligent scheduling decisions, real-time monitoring, and observability tools were used. This includes the integration of Prometheus for system-level metrics collection, Grafana for data visualization, and iperf for bandwidth testing between nodes. These tools were chosen not only for their open-source nature but also for their compatibility with ARM architectures.

5.2.1 Operating System

All Raspberry Pi devices are installed with the operating system Ubuntu 24.04 for ARM, from the Raspberry Pi Imager. This operating system is compatible with K3s and supports essential development and networking tools. The operating system was downloaded onto micro SD cards using the official Raspberry Pi Imager, allowing for fast, consistent deployment across all nodes.
Official Documentation: Raspberry Pi Imager

5.2.2 Prometheus

Prometheus was used as the primary tool for metric collection within the Kubernetes cluster. It is an open-source monitoring and alerting tool designed specifically for time-series data. In this project, Prometheus was used to scrape the metrics from each Raspberry Pi node using the Node Exporter. Some of the metrics collected are system load, CPU usage, and temperature.
Official Documentation: Prometheus

5.2.3 iPerf

To check the network performance and to add the real-time bandwidth availability into the scheduling decisions, iperf3 was installed on each node. iperf is a widely-used network tool, and it can measure maximum TCP and UDP bandwidth between any two devices. These metrics were collected at regular intervals between all device pairs in the cluster.
Official Documentation: iperf

5.2.4 Grafana

For the visualization of the cluster metrics, Grafana was used alongside Prometheus. It gets the data directly from Prometheus, and it visualizes metrics such as energy metrics, bandwidth metrics, and CPU utilization.
Official Documentation: grafana

5.3 Development of the Custom Scheduler

The main output of the implementation was a fully functional custom scheduler developed using the Kubernetes Scheduling Framework. Written in the Go programming language, the scheduler uses Kubernetes' plugin architecture to compute node scores dynamically. These scores were based on real-time data for bandwidth availability and energy usage, in addition to standard metrics like CPU and memory. The scheduler used all these metrics for the score to rank nodes, ensuring that workloads were assigned to the most efficient node under current conditions.
Official Documentation: Scheduling Framework

5.4 Techniques and Algorithms

The development of the custom Kubernetes scheduler requires the integration of several techniques and decision-making algorithms to improve the default scheduling behavior. While the default Kubernetes scheduler mainly considers CPU and memory availability for pod placement, this project expands the scheduling criteria to include real-time bandwidth and energy efficiency. These additions are very useful in fog and edge computing environments. This scheduler uses a multi-metric scoring algorithm. This scoring system evaluates all available nodes based on a weighted sum of normalized metrics. Each metric, CPU usage, available memory, bandwidth, and energy consumption, is assigned a relative weight based on its importance to the application.

The scheduler filters out nodes that do not meet the basic requirements, as in the default scheduler. Among the feasible nodes, a score is calculated using the formula:

$$\text{NodeScore} = \alpha \cdot C + \beta \cdot M + \gamma \cdot B + \delta \cdot \left(\frac{1}{E}\right)$$

where:	C	= Normalized CPU availability
	M	= Normalized Memory availability
	B	= Normalized Bandwidth availability
	E	= Node Energy usage
	$\alpha, \beta, \gamma, \delta$	= Weight coefficients for each metric

5.5 Metrics Integration and Visualization

A significant portion of the implementation effort involved integrating real-time node metrics into the scheduling logic. Prometheus collected energy metrics, while iPerf measured bandwidth between nodes. These metrics were visualized using Grafana dashboards. With the help of these dashboards, we can check the resource consumption and overall system behavior.

5.6 Output

Once the custom scheduler was developed and deployed, its effectiveness was evaluated against the default scheduler under identical workloads. Metrics such as scheduling time, bandwidth efficiency, and energy distribution were recorded and compared. The results demonstrated that the custom scheduler significantly improved workload distribution, reduced communication overhead, and utilized energy more effectively. These improvements were particularly evident in bandwidth-intensive scenarios, where intelligent node selection resulted in improved performance.

6 Evaluation

6.1 Overview

This section presents a comprehensive evaluation of the proposed custom Kubernetes scheduler that integrates bandwidth and energy efficiency metrics for improved workload

```
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  5m50s  default-scheduler  Successfully assigned default/nginx-deployment-default-7989c6489c-2g694 to abhinav-worke
r-1
  Normal  Pulling    5m49s  kubelet            Pulling image "nginx:latest"
  Normal  Pulled     5m48s  kubelet            Successfully pulled image "nginx:latest" in 1.276s (1.276s including waiting). Image siz
e: 68855984 bytes.
  Normal  Created    5m48s  kubelet            Created container: nginx-cont
  Normal  Started    5m48s  kubelet            Started container nginx-cont
```

Figure 3: Default Scheduler Scheduling Time

```
Events:
  Type    Reason     Age   From     Message
  ----    ------     ----  ----     -------
  Normal  Scheduled  27s   Peaks    Successfully assigned default/nginx-deployment-peaks-557f4cbc4f-fjvfb to abhinav-worker-1
  Normal  Pulling    27s   kubelet  Pulling image "nginx:latest"
  Normal  Pulled     26s   kubelet  Successfully pulled image "nginx:latest" in 1.017s (1.017s including waiting). Image size: 68855984
 bytes.
  Normal  Created    26s   kubelet  Created container: nginx-cont
  Normal  Started    26s   kubelet  Started container nginx-cont
```

Figure 4: Custom Scheduler Scheduling Time

placement in an edge computing environment using Raspberry Pi clusters. The main aim of the evaluation was to determine whether the custom scheduler improves workload distribution efficiency, reduces communication overhead, and enhances energy utilization compared to the default Kubernetes scheduler. Multiple experiments were conducted to assess the performance of the custom scheduler in real-world use cases.

6.2 Evaluation Metrics

To accurately assess the performance of the custom Kubernetes scheduler, several key evaluation metrics were identified. All these metrics are chosen based on the scheduler's intended improvements in workload efficiency, network performance, and energy efficiency. The metrics chosen are:

Scheduling Time: Measured the time taken from pod creation to pod running state, indicating how efficiently the scheduler handles incoming workloads.

Bandwidth: This helps in identifying the scheduler correctly favored nodes with higher available bandwidth or not, especially for workloads requiring significant data transfer.

Energy Usage: Evaluated the thermal and CPU load distribution across nodes as a proxy for energy usage, which is very important in edge environments as these lack active cooling.

CPU Load: Tested the scheduler's responsiveness to dynamic changes such as increased CPU stress on specific nodes.

Metrics were collected using a combination of Prometheus, Node Exporter, and iperf, while Grafana dashboards were used to visualize patterns. These metrics validated the theoretical design of the custom scheduler and its practical benefits compared to the default scheduler.

$$\text{NodeScore} = \alpha \cdot C + \beta \cdot M + \gamma \cdot B + \delta \cdot \left(\frac{1}{E}\right)$$

6.2.1 Experiment 1: Pods Scheduling Time

In the first experiment, it demonstrates the average scheduling time between the default Kubernetes scheduler and the custom scheduler. A workload consisting of identical pods was deployed repeatedly under both schedulers. The scheduling time for each pod, from

the moment of request to the container creation state and the pod running state, can be seen here.

6.2.2 Experiment 2: Pods Bandwidth

In this experiment, the pod bandwidth is evaluated by deploying the deployment manifest using both the custom and default schedulers. There is just a minute difference between custom scheduler deployments and default scheduler deployments. From this, we can conclude that this custom scheduler can be deployed into the real world. All the graphs are from the Grafana dashboards.

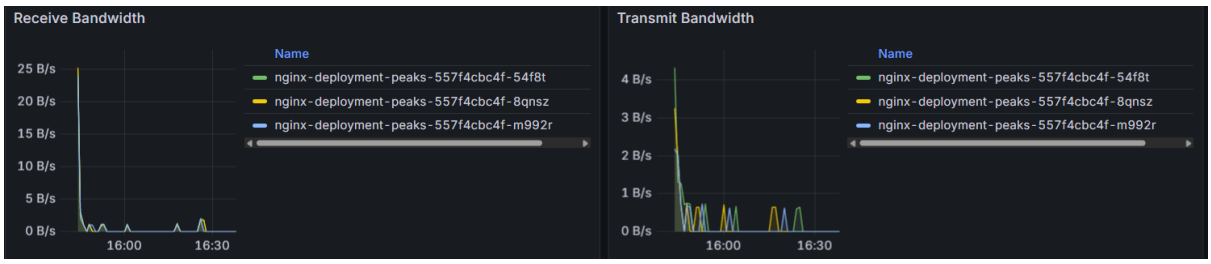

Figure 5: Custom Scheduler Deployment

Figure 6: Default Scheduler Deployment

6.2.3 Experiment 3: Energy Efficiency

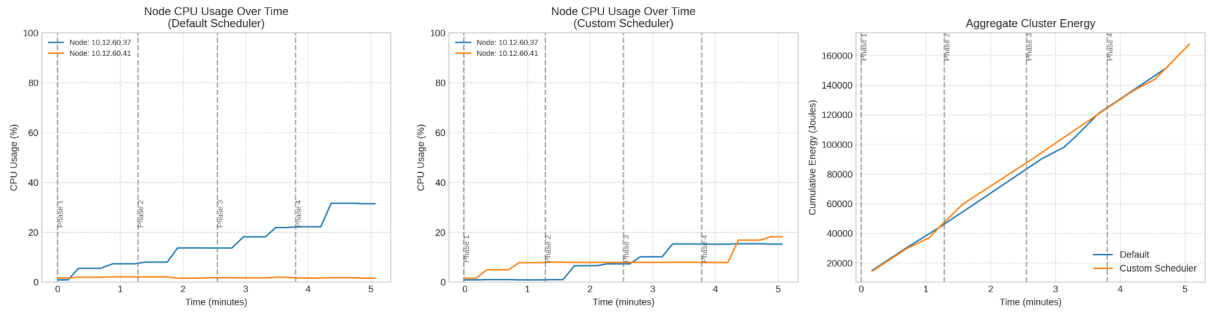

Figure 7: Energy Metrics Graphs

This experiment analyzed CPU temperature and energy draw using Prometheus Node Exporter. Pods with identical CPU usage profiles were deployed over an extended period from 5 minutes to 30 minutes under both schedulers. Some of the metrics evaluated are temperature distribution, energy consumption through CPU load, and temperature. All

the worker nodes have been evaluated using these metrics to find the best node for the workload placement.

6.2.4 Experiment 4: Real Time Adaptability

In this experiment, a mixed workload scenario was introduced, where network congestion was simulated on one node midway through the deployment using stress-ng; this stress has been simulated manually. The goal was to see if the custom scheduler responded dynamically to changing conditions when increasing the stress on one worker node. Here, the default scheduler continued assigning pods to the congested node due to a lack of awareness.

Figure 8: Power Usage

```
nginx-deployment-peaks-557f4cbc4f-8jkpv   1/1   Running   0   23s   10.42.2.32   abhinav-worker-1   <none>   <none>
nginx-deployment-peaks-557f4cbc4f-bndck   1/1   Running   0   23s   10.42.2.33   abhinav-worker-1   <none>   <none>
nginx-deployment-peaks-557f4cbc4f-g6lns   1/1   Running   0   23s   10.42.4.224  abhinav-worker-2   <none>   <none>
nginx-deployment-peaks-557f4cbc4f-k8ps7   1/1   Running   0   23s   10.42.4.228  abhinav-worker-2   <none>   <none>
nginx-deployment-peaks-557f4cbc4f-kfxnh   1/1   Running   0   23s   10.42.4.226  abhinav-worker-2   <none>   <none>
nginx-deployment-peaks-557f4cbc4f-l4mrt   1/1   Running   0   23s   10.42.4.225  abhinav-worker-2   <none>   <none>
nginx-deployment-peaks-557f4cbc4f-m7h2x   1/1   Running   0   23s   10.42.4.227  abhinav-worker-2   <none>   <none>
nginx-deployment-peaks-557f4cbc4f-xgwxs   1/1   Running   0   23s   10.42.4.229  abhinav-worker-2   <none>   <none>
nginx-deployment-peaks-557f4cbc4f-zb4b7   1/1   Running   0   23s   10.42.4.223  abhinav-worker-2   <none>   <none>
```

(a) Custom Scheduler

```
abhinav@abhinav-master:~/k8s$ kubectl get po -o wide
NAME                                       READY   STATUS    RESTARTS   AGE     IP           NODE               NOMINATED NODE   READINESS GATES
cpu-stress-default-phase2-v5rhv            1/1     Running   0          7m49s   10.42.2.20   abhinav-worker-1   <none>           <none>
cpu-stress-default-phase3-fdxmn            1/1     Running   0          6m35s   10.42.2.21   abhinav-worker-1   <none>           <none>
cpu-stress-default-phase4-5j2dl            1/1     Running   0          6m32s   10.42.2.22   abhinav-worker-1   <none>           <none>
nginx-deployment-default-7989c6489c-426sf  1/1     Running   0          2m55s   10.42.2.23   abhinav-worker-1   <none>           <none>
nginx-deployment-default-7989c6489c-4xmwg  1/1     Running   0          2m55s   10.42.2.27   abhinav-worker-1   <none>           <none>
nginx-deployment-default-7989c6489c-785qc  1/1     Running   0          2m55s   10.42.4.217  abhinav-worker-2   <none>           <none>
nginx-deployment-default-7989c6489c-78f5x  1/1     Running   0          2m55s   10.42.4.214  abhinav-worker-2   <none>           <none>
nginx-deployment-default-7989c6489c-ffzw8  1/1     Running   0          2m55s   10.42.2.24   abhinav-worker-1   <none>           <none>
nginx-deployment-default-7989c6489c-jzdhx  1/1     Running   0          2m55s   10.42.4.216  abhinav-worker-2   <none>           <none>
nginx-deployment-default-7989c6489c-tblqq  1/1     Running   0          2m55s   10.42.4.215  abhinav-worker-2   <none>           <none>
nginx-deployment-default-7989c6489c-thz67  1/1     Running   0          2m55s   10.42.2.26   abhinav-worker-1   <none>           <none>
nginx-deployment-default-7989c6489c-z89vx  1/1     Running   0          2m55s   10.42.2.25   abhinav-worker-1   <none>           <none>
```

(b) Default Scheduler

Figure 9: Comparison between schedulers: (a) Custom vs (b) Default

6.3 Discussion

All these experiments collectively demonstrate that the custom scheduler substantially improves upon the Kubernetes default scheduling when used in edge environments. This project supports that incorporating bandwidth and energy metrics can optimize workload distribution for latency-sensitive, energy-constrained, and bandwidth-intensive applications. The custom scheduler not only improved pod scheduling but also showed significant intelligence in reacting to real-time metric changes, something the default scheduler is unable to achieve. These findings align with prior research, such as KEIDS, which emphasized energy and network-awareness, validating the relevance of the proposed solution.

However, the evaluation is not without limitations. For example, energy usage was estimated from proxy metrics such as CPU temperature and load due to the lack of direct

power sensors on Raspberry Pi devices. Also, the experiments were conducted in a controlled lab environment. Further testing is required in a production environment in a fog computing setup that would strengthen the adaptability of this project in the real world. Overall, the evaluation supports the core objectives of this study and opens pathways for further improvements in resource-aware orchestration in fog and edge environments.

7 Conclusion and Future Work

This research is done to address the following question: Can a custom Kubernetes scheduler, which is incorporated with energy efficiency metrics, outperform the default scheduler in terms of communication overhead and workload distribution efficiency in an edge environment?

To address this, a lightweight Kubernetes cluster was deployed using Raspberry Pi devices, and a custom scheduler was developed using the Kubernetes Scheduling Framework. The scheduler uses real-time metrics collected from Prometheus and iPerf to get bandwidth and energy usage. These metrics were then used to intelligently rank nodes for pod placement, thereby improving the default scheduling logic, which primarily considers CPU and memory. The primary objectives of the study were to design and implement a custom scheduler that supports bandwidth and energy-aware scheduling, to deploy and monitor a Kubernetes cluster using low-cost edge hardware, and to evaluate the performance of the custom scheduler against the default scheduler. These objectives were successfully achieved. The custom scheduler demonstrated clear improvements across key dimensions, including scheduling latency, energy-balanced node usage, and responsiveness to real-time network changes. These results support that integrating environmental and resource-aware metrics leads to more efficient workload orchestration in edge computing environments. The implementation proved to be not only functional but also cost-effective and relevant to new trends in fog computing.

The current project opens several promising implementations for future research and potential commercial applications: future work could involve integrating real-time energy sensors to gather accurate power consumption data per node. This would enable the scheduler to support not only energy-aware scheduling but also sustainability-aware workload decisions that are important in renewable edge environments, and use external networking hardware that collects the real-time bandwidth of the nodes to decouple the current architecture. For scalability, the architecture remains feasible if the project is extended to larger clusters, though the metric collection overhead and network congestion may increase. For example, in a smart factory IoT system, a cluster of 10 to 15 Raspberry Pis could support multiple sensor-driven applications, but ensuring up-to-date metrics becomes very important. Another extension would be to integrate machine learning into the scheduling decisions for better placements. The machine learning model will be trained on historical Prometheus data to predict node performance and network behavior, and a predictive scheduler could proactively allocate workloads, reducing latency and maximizing resource utilization.

References

Arunan, S., Amarasinghe, G. and Perera, I. (2023). Cost-optimized scheduling for microservices in kubernetes, *2023 IEEE International Conference on Cloud Computing*

Technology and Science (CloudCom), IEEE, pp. 131–138.

Canova, S. (2024). Scheduling optimization in kubernetes.

Carrión, C. (2022). Kubernetes scheduling: Taxonomy, ongoing issues and challenges, *ACM Computing Surveys* **55**(7): 1–37.

Chandra, M. (2023). *Effective memory utilization using custom scheduler in kubernetes*, PhD thesis, Dublin, National College of Ireland.

Donca, I.-C., Stan, O. P., Misaros, M., Stan, A. and Miclea, L. (2024). Comprehensive security for iot devices with kubernetes and raspberry pi cluster, *Electronics* **13**(9): 1613.

Eidenbenz, R., Pignolet, Y.-A. and Ryser, A. (2020). Latency-aware industrial fog application orchestration with kubernetes, *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 164–171.

Füstös, F., Péter, K., László, N.-P., Mátis, S.-G., Szabó, Z. and Sulyok, C. (2022). Managing a kubernetes cluster on raspberry pi devices, *2022 IEEE 20th Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*, pp. 133–138.

Gizinski, T. and Cao, X. (2022). Design, implementation and performance of an edge computing prototype using raspberry pis, *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0592–0601.

Haja, D., Szalay, M., Sonkoly, B., Pongracz, G. and Toka, L. (2019). Sharpening kubernetes for the edge, *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, pp. 136–137.

Kang, B., Jeong, J. and Choo, H. (2021). Docker swarm and kubernetes containers for smart home gateway, *IT Professional* **23**(4): 75–80.

Kaur, K., Garg, S., Kaddoum, G., Ahmed, S. H. and Atiquzzaman, M. (2020). Keids: Kubernetes-based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem, *IEEE Internet of Things Journal* **7**(5): 4228–4237.

Kayal, P. (2020). Kubernetes in fog computing: Feasibility demonstration, limitations and improvement scope : Invited paper, *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pp. 1–6.

Li, H., Shen, J., Zheng, L., Cui, Y. and Mao, Z. (2023). Cost-efficient scheduling algorithms based on beetle antennae search for containerized applications in kubernetes clouds, *The Journal of Supercomputing* **79**(9): 10300–10334.

Schnider, M. and Hutter, P. (2024). *Kubernetes Pod Scheduling based on Energy efficient Metrics*, PhD thesis, OST Ostschweizer Fachhochschule.

Senjab, K., Abbas, S., Ahmed, N. and Khan, A. u. R. (2023). A survey of kubernetes scheduling algorithms, *Journal of Cloud Computing* **12**(1): 87.