International Journal of Technology

http://ijtech.eng.ui.ac.id

Research Article

Digital Twin for Indoor Farming Using Kubernetes Container-Based Application with Orchestration-Centric Performance Modeling

Ramot Lubis[1], Augie Widyotriatmo[2], Endra Joelianto[2*]

[1]Doctoral Program of Engineering Physics, Faculty of Industrial Technology, Institut Teknologi Bandung, Bandung, 40132, Indonesia

[2]Instrumentation, Control, and Automation Research Group, Institut Teknologi Bandung, Bandung, 40132, Indonesia

[*]Corresponding author: ejoel@itb.ac.id; Tel.: +6222-2504424; Fax: +6222-2506281

Abstract: Indoor farming requires scalable, reliable, and low-latency monitoring and control to maintain optimal crop growth environmental conditions. Digital Twin (DT) technology offers a promising paradigm for virtually representing and managing physical farming systems. However, practical DT implementations for indoor agriculture often lack analytically grounded scalability models for edge-orchestrated DT systems. This study presents DTFarming, a microservices-based digital twin monitoring and control system for indoor farming deployed as a containerized service orchestrated using Kubernetes. Message Queuing Telemetry Transport (MQTT) is employed for event-driven telemetry exchange. This study proposes an analytical scalability model that relates sensor density to workload arrival rate, pod population, resource cost, memory saturation, and Kubernetes scheduling pressure. Experimental validation across three load scenarios (4, 20, and 100 sensors) confirms the predictive capability of the proposed model. The results indicate that the telemetry throughput scales linearly with minimal CPU overhead (<25%). However, memory is the primary bottleneck, capping scalability at ∼110 pods on an 8GB edge node. This experimental saturation point aligns almost perfectly with our analytical model (≈112 pods), exhibiting a deviation of less than 2%. As deployments approach this limit, Kubernetes scheduling latency rises from 0.4 s to 2.1 s—a 5.25 times increase in the Scheduling Pressure Index (SPI)—while network latency and packet loss remain negligible. These findings confirm that orchestration overhead, memory constraints, dominates performance degradation rather than communication limits. The close agreement between model predictions and measurements confirms the model's validity and offers practical guidance for sizing container-orchestrated DT platforms on constrained edge nodes.

Keywords: Digital twin; Edge computing; Indoor farming; Internet of things (IoT); Microservices

1. Introduction

Indoor farming represents a sustainable agricultural approach based on controlled-environment agriculture (CEA), where environmental conditions are precisely regulated to optimize crop growth. However, achieving optimal productivity requires precise monitoring and control of environmental parameters such as temperature, humidity, CO_2 levels, and light intensity. Digitalization can improve these systems through better monitoring, data analysis, and decision support.

The Internet of Things (IoT), along with cloud and edge computing systems, enables seamless communication among software platforms and diverse physical entities in digital agriculture (Pylianidis et al., 2021). To address performance, Kristiani et al. (2021) utilized microservices to reduce latency, while others optimized cloud-edge interactions through event-driven architec-

https://doi.org/10.14716/ijtech.v17i4.8301
Received December 2025; Revised February 2026; Accepted May 2026

tures, such as MQTT, to minimize resource consumption and improve data exchange efficiency. Alam et al. (2018) architecturally established this approach, and Li et al. (2023a) refined it for lightweight deployment in constrained environments. Kubernetes-based architectures have been employed for real-time decision-making and predictive modeling (Fakeye et al., 2024). These technologies are now prevalent across agricultural domains: intelligent irrigation leveraging sensor feedback (Boursianis et al., 2021), climate control in vertical farms (Li et al., 2023b), and soil monitoring (Gopalakrishnan et al., 2021). Furthermore, recent research has pivoted toward Digital Twins (DT), emphasizing maturity through model-based systems engineering (Madni et al., 2019) and novel architectures specifically for smart farming (Pylianidis et al., 2021).

Kubernetes (K8s) is a widely adopted orchestration platform that automates deployment, scaling, and management of containerized applications (Böhm & Wirtz, 2021). Kubernetes is a large platform; hence, it has complexities and resource consumption. However, lightweight alternatives have emerged, including MicroK8s and K3s, which are optimized for edge computing and IoT applications (Kjorveziroski & Filiposka, 2022). KubeEdge (KubeEdge, 2024) is an innovation around microservices at the edge.

Another study (Koziolek & Eskandani, 2023) benchmarked different Kubernetes distributions and found that k3s and k0s generally exhibit lower control-plane overhead and faster orchestration responsiveness. However, (Böhm & Wirtz, 2021) suggested that MicroK8s exhibited higher resource consumption and longer operation times. This study (Usman et al., 2024) evaluated full-fledged Kubernetes, K3s, and MicroK8s for serverless edge workloads. MicroK8s includes a complete Kubernetes environment with built-in features, such as DNS, storage, and monitoring, making it a more self-contained distribution than K3s (Usman et al., 2024). MicroK8s performs better on Ubuntu-based systems due to its optimized integration with the ecosystem of Canonical.

A recent study focusing on the iEC 61499 toward microservice architectures in distributed control systems represents a paradigm shift in industrial automation. Dai et al. (2023) demonstrated that orchestration methods and deployment procedures for OT–IT achieve significant performance improvements. Their work established containerized function blocks.

A DT framework (Grieves, 2019) enables real-time simulation and decision-making through a virtual representation of physical systems. DT enables the decoupling of physical flows from planning and control, allowing farmers to remotely manage operations based on near real-time digital information (Verdouw et al., 2021). The DT approach has evolved, with Tao et al. (2019) proposing a five-stage framework for DT maturity in industrial contexts, which was later extended by Jeong et al. (2022), who mapped specific technology elements to each implementation stage. This work addresses the first three stages, namely, virtualization, synchronization, and modeling simulation, as identified in Nasirahmadi and Hensel (2022).

Table 1 Comparison of Kubernetes edge-orchestrated studies

Study	The Digital Twin in Agriculture	Edge Kubernetes deployment	The analytical Scalability Model	Orchestration-Layer Metrics	Analysis of Scalability Phase	Memory-Saturation Prediction
(Pylianidis et al., 2021)	✓	×	×	×	×	×
(Nasirahmadi & Hensel, 2022)	✓	×	×	×	×	×
(Verdouw et al., 2021)	✓	×	×	×	×	×
(Fakeye et al., 2024)	✓	✓	×	×	×	×
(Pan et al., 2022)	×	✓	×	×	×	×
(Böhm & Wirtz, 2021)	×	✓	×	✓	×	×
(Koziolek & Eskandani, 2023)	×	✓	×	✓	×	×
(Usman et al., 2024)	×	✓	×	✓	×	×
(Dai et al., 2023)	×	✓	×	✓	×	×

Table 1 shows that prior studies can be grouped into three areas. The first area focuses on agricultural DT concepts, data integration, and farm-level decision support, where works such as Verdouw et al. (2021), Nasirahmadi and Hensel (2022), and Pylianidis et al. (2021)

clarify DT roles in smart farming but do not quantify orchestration layer scalability or edge container density. The second area evaluates lightweight Kubernetes distributions for edge computing. Studies comparing MicroK8s, K3s, k0s, MicroShift, and other Kubernetes variants focus mainly on control-plane throughput, resource footprint, latency, and general platform suitability rather than DT-agent container density modeling. For example, Koziolek and Eskandani (2023) reported that k3s and k0s show slightly higher control-plane throughput, whereas MicroShift shows strong data-plane throughput, but their study is not framed around Digital Twin density or memory-bound saturation. Similarly, recent comparative studies by Pan et al. (2022) and Usman et al. (2024) evaluated resource consumption, latency, throughput, and security across lightweight Kubernetes variants but did not derive a DT agent predictive density model. The third area investigates containerized IoT or industrial microservices; however, orchestration is generally treated as an implementation layer rather than a measurable scalability constraint. For instance, Dai et al. (2023) investigated microservice orchestration for industrial IoT edge systems, while Fakeye et al. (2024) discussed distributed DT service deployment and containerized orchestration frameworks without modeling orchestration-layer saturation behavior. None of the reviewed studies derive a memory-bound predictive density model for containerized Digital Twin agents deployed on resource-constrained edge nodes.

Therefore, the gap addressed in this study is not the absence of DT architectures or Kubernetes benchmarks but the lack of an orchestration-centric model that links DT agent container density, memory saturation, and scheduling latency on constrained edge nodes. The central hypothesis comprises two coupled claims: (i) under resource-constrained edge conditions, increasing DT agent container density induces approximately linear growth in cumulative memory consumption because each containerized agent carries a near-constant per-pod memory cost; and (ii) as memory utilization approaches the node's saturation threshold, the Kubernetes orchestration layer—rather than the network or CPU sub-system—becomes the dominant source of end-to-end responsiveness degradation, producing a nonlinear rise in scheduling latency that marks the transition from a nominal to a saturation-limited phase.

This study employs deterministic telemetry generation to isolate orchestration effects, thereby minimizing stochastic variability on the input side. This controlled design enables the nonlinear scheduling latency growth to be attributed to intrinsic resource contention within the Kubernetes control plane rather than to burst-driven traffic fluctuations. The resulting model characterizes the internal orchestration constraints governing the density scaling of DT.

This study is conducted within the broader development of the DTFarming platform (Lubis et al., 2024), a microservices-based digital twin system for indoor precision farming deployed on edge infrastructure. The microservices architecture is adopted to support modularity and distributed control evolution, enabling the systematical evaluation of density-driven scalability behavior in resource-constrained environments.

The primary contributions are fourfold: (i) the formulation of an analytical scalability model relating sensor density to workload rate, pod population, per-agent resource cost, cumulative memory consumption, and scheduling latency; (ii) the proposal of the Scheduling Pressure Index (SPI), a normalized metric quantifying orchestration-layer stress under increasing DT density; (iii) the identification and experimental confirmation of two scalability phases—nominal linear growth and saturation—through deployments with 4, 20, and 100 sensors; and (iv) a latency decomposition demonstrating that orchestration delay dominates network latency at high density, suggesting that orchestration-layer memory pressure may dominate scalability constraints within the evaluated deployment regime in IoT-driven DT systems.

2. Methods

The proposed DTFarming architecture integrates farming physical devices (sensors and actuators), edge computing devices (ESP32, Arduino, Raspberry Pi, etc.), cloud infrastructure (Kubernetes, Storage, etc.), and other software such as monitoring tools (Prometheus, Grafana), MongoDB No-SQL database, MQTT event-driven messaging. The architecture enables seamless

communication across layers through MQTT and HTTP REST protocols (Turnip et al., 2023). Furthermore, Figure 1 illustrates the key components of the DTFarming system.

This model shows how physical indoor farming is regulated with the DTFarming software in terms of monitoring and control. The monitoring system gathers environmental data and refines it using estimator models to improve accuracy. Observation data are crucial to continue with control processing so that DTFarming could dictate optimal control of the agriculture environment based on the control model and knowledge base.

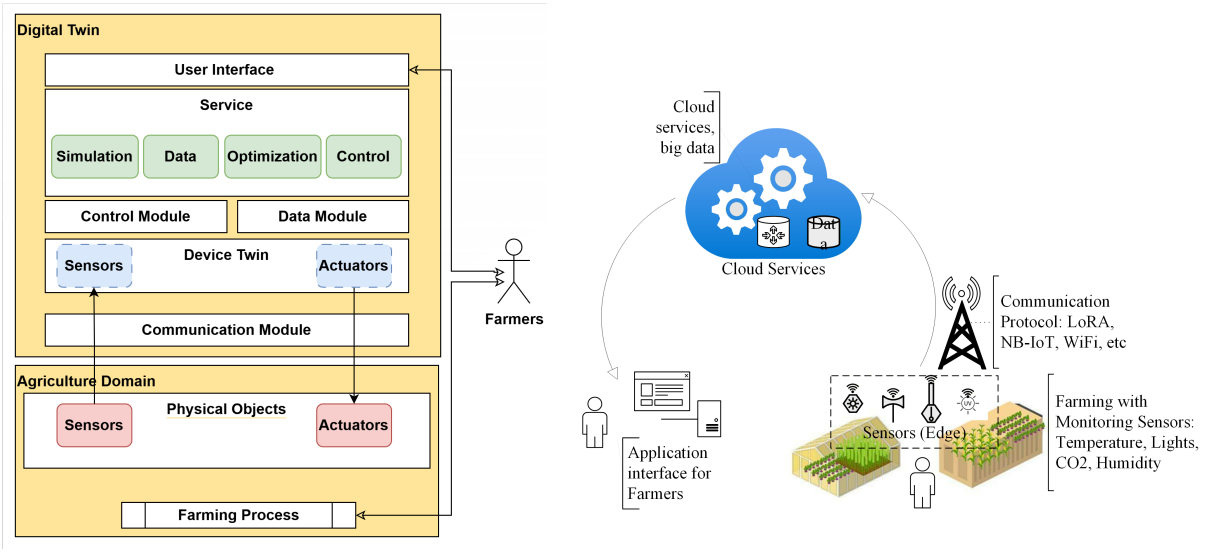


Figure 1 (a) DTFarming component model and (b) physical environment

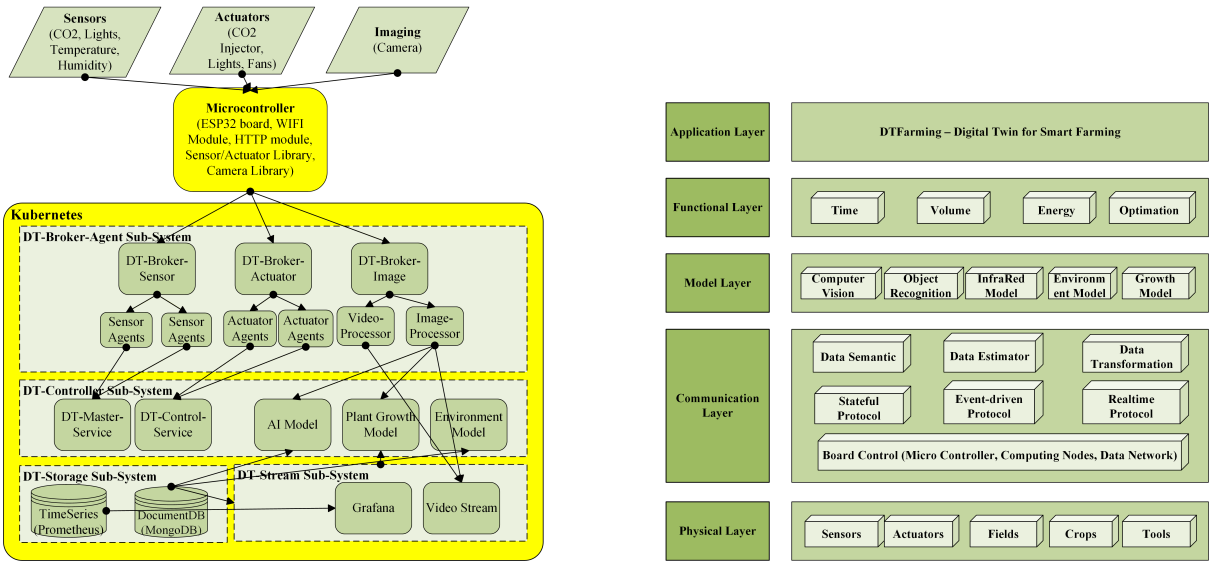
The communication model typically employs protocols such as HTTP, SOAP, COAP, and MQTT, with HTTP Rest being the most widely adopted for its ease of use and standardization, facilitating high levels of adoption.

Figure 2 illustrates the DT model showing how physical devices, such as sensors or actuators, are represented in a containerized service, referred to as DT container agents. These containers are unified through a master hub known as Digital Twin Central (DTC), which serves as the interface to other modules/components or provides access to the user or presentation layer (Lubis et al., 2024).

Sensor Broker is a middleware service for unifying sensor data formats. It adopts an adapter pattern (Burns & Oppenheimer, 2016) to standardize the communication and data format from the physical devices. The broker acts as an intermediary between physical devices, service agents, and other system components. It utilizes both HTTP and MQTT protocols for data exchange (Turnip et al., 2023).

MongoDB is a document-oriented database system optimized for internet of Things applications, particularly those generating time-series data. Historical sensor readings are stored for subsequent analysis, control operations, and front-end visualization. Each component runs independently in a containerized service, enabling dynamic scalability and isolated execution and interservice communication through the HTTP and MQTT protocols.

Figure 3 shows the sequence diagram to provide a visual representation of the interactions among the system components. The master service orchestrates sensor and control agents, ensures data consistency, and manages interactions with persistent data storage. It also mediates data retrieval from the storage system for the frontend layer. The MQTT Service manages communications between sensor agents and the master service using a publish-subscribe architecture, ensuring reliable, event-driven data exchange (Gupta et al., 2018).



(a) (b)

Figure 2 (a) DTFarming Runtime Architecture (b) Conceptual Layered Architecture

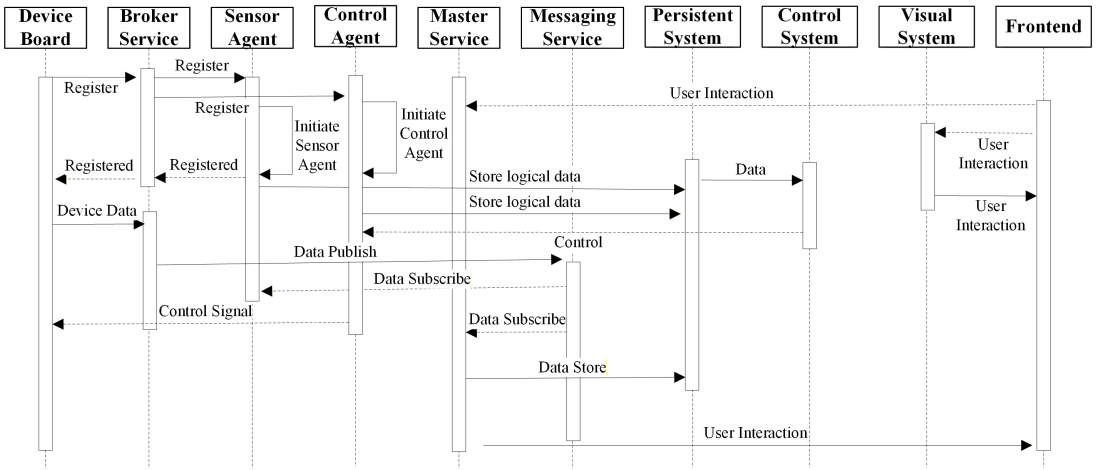


Figure 3 Sequence diagram of the DTFarming component

2.1 Definition of model performance metrics

To quantitatively evaluate the scalability and efficiency of the proposed DTFarming system, a set of metrics covering orchestration behavior, resource utilization, and network performance is defined. Based on the workload, resource, and scheduling models defined in Eqs. (1)–(12), the DTFarming system's scalability behavior can be summarized as follows.

As the number of sensors increases, the message arrival rate grows linearly, leading to a proportional increase in the number of containerized DT agent pods. While the per-agent CPU cost remains approximately constant, the cumulative memory consumption grows linearly with N_s, eventually approaching the memory saturation threshold.

Once memory pressure increases beyond this threshold, Kubernetes scheduling latency rises sharply, as reflected by an increase in the scheduling pressure index. This transition marks the shift of the system from a nominal operating phase to a saturation-limited phase. Therefore, under peak load conditions, end-to-end DT responsiveness is dominated by orchestration delays rather than network latency.

Kubernetes scheduling latency measures the orchestration layer's responsiveness when deploying DT agents. The elapsed time between a pod creation request and the pod reaching the running state is defined as follows:

$$T_s = t_{running} - t_{created} \tag{1}$$

Where $t_{created}$ is the timestamp when the pod specification is submitted to the Kubernetes API server, and $t_{running}$ is the timestamp when the pod transitions to the *running* phase.

The average scheduling latency for a given scenario is computed as follows:

$$\bar{T}_s = \frac{1}{N_p} \sum_{i=1}^{N_p} T_{s,i} \tag{2}$$

where N_p is the number of scheduled pods during the experiment.

SPI is defined as the ratio of the mean scheduling latency at density N_s relative to the baseline scenario (4 sensor containers). Kubernetes scheduling pressure index (SPI) is denoted as follows:

$$SPI(N_s) = \frac{T_s(N_s)}{T_s(N_s = 4)} \tag{3}$$

Standard error:

$$SE_e = \frac{SD_e}{\sqrt{N_e}}$$

Event-level 95% CI:

$$CI_{95\%,e} = \bar{T}_s \pm t_{0.975,N_e-1} \cdot SE_e$$

where

N_s is the number of sensor containers (and therefore DT agent pods);

$T_s(N_s)$ is the average scheduling latency under the load;

$T_s(N_s = 4)$ is the baseline scheduling latency (in this case, 4 sensor containers are used as the baseline);

$t_{0.975,N_e-1} \approx 1.96$.

The SPI metric is motivated by prior Kubernetes orchestration-performance literature, where scheduler efficiency and system saturation are commonly assessed through latency, throughput, resource utilization, interference, and workload responsiveness metrics. Existing surveys show that Kubernetes scheduling studies frequently evaluate resource utilization, latency reduction, load balancing, autoscaling behavior, and application-level performance as core scheduling objectives (Senjab et al., 2023). More recent orchestration studies further demonstrate that established low-level metrics, such as scheduling latency for quantifying co-located workload interference (Li et al., 2023b) and end-to-end user-perceived latency for latency-aware service placement at the edge, can be used to derive higher-level indicators (Centofanti et al., 2023). Similarly, lightweight Kubernetes benchmarking studies compare MicroK8s, k3s, k0s, and MicroShift using CPU and memory utilization, pod creation throughput, pod creation latency, deployment/pod operation latency, and data-plane throughput/latency, showing that orchestration performance is multidimensional and may differ between control-plane and data-plane behavior (Kjorveziroski & Filiposka, 2022; Koziolek & Eskandani, 2023). Based on this foundation, SPI is introduced not as a replacement for established Kubernetes metrics but as a normalized orchestration-pressure indicator that expresses scheduling amplification relative to a baseline Digital Twin deployment, especially under increasing DT-agent container density where resource saturation and pod scheduling delay jointly influence edge scalability.

The CPU use quantifies the processing overhead introduced by the DT agents and core services. For a service i, CPU usage is measured in millepores (mCPU) and averaged over the experiment duration T:

$$CPU_j = \frac{1}{T} \int_0^T U_j(t) dt \tag{4}$$

where $u_j(t)$ denotes instantaneous CPU usage reported by Kubernetes metrics.

Memory use captures each service's runtime memory footprint. For service j, average memory usage is defined as follows:

$$MEM_j = \frac{1}{T}\int_0^T m_j(t)\,dt \tag{5}$$

where $m_j(t)$ represents the resident memory consumption at time t.

The network throughput evaluates the MQTT-based communication capacity between DT agents and the master service. It is defined as the number of successfully delivered messages per second:

$$\text{Throughput} = \frac{N_m}{T} \tag{6}$$

where N_m is the total number of MQTT messages received during the time interval T.

End-to-end message latency L is measured as follows:

$$L = t_{recv} - t_{send} \tag{7}$$

The packet loss rate is defined as

$$PLR = \frac{N_{sent} - N_{received}}{N_{sent}} \times 100\% \tag{8}$$

where N_{sent} and $N_{received}$ denote the number of messages transmitted and received successfully, respectively.

Number of container pods running in the cluster consuming physical resources:

$$N_p = N_s + N_b + N_m + N_d + N_o \tag{9}$$

where:
N_s : sensor agent pods
N_b : broker pods
N_m : master service pods
N_d : database pods
N_o : observability pods

The Per-Agent Resource Cost Model:

$$C_{mem} = \frac{1}{N_s}\sum_{i=1}^{N_s} MEM_i \tag{10}$$

$$C_{cpu} = \frac{1}{N_s}\sum_{i=1}^{N_s} CPU_i$$

Threshold of Memory Saturation

$$N_s^{max} = \left[\frac{|M_{total} - M_{base}|}{C_{mem}}\right] \tag{11}$$

where:
M_{total} : available RAM
M_{base} : OS + Kubernetes overhead
DT responsiveness latency is defined as follows:

$$L_{DT} = L_{net} + L_{proc} + L_{sched} \tag{12}$$

where:

L_{net} : telemetry transmission delay (measured through timestamps / broker)
L_{proc} : processing + persistence time at the broker/master/DB
L_{sched} : orchestration delay when new pods are created
This study emphasizes L_{net} and L_{sched} as dominant contributors to responsiveness in elastic, edge-deployed DTFarming microservices.

Based on the observed system behavior, two scaling phases are identified:
(i) a nominal phase ($N_s \leq 20$), where resource usage and scheduling latency increase approximately linearly with the sensor count; and
(ii) a saturation phase ($N_s \geq 80$), where memory constraints dominate and the scheduling latency increases nonlinearly.

2.2 Architectural Model

The DTFarming platform adopts a microservice architecture that decouples physical control from virtual orchestration across three planes: physical, orchestration, and application. Containerized digital twin agent mirrors each physical sensor i, forming a device-twin isomorphism that provides strong fault isolation and independent lifecycle management. State consistency is maintained through event-sourced synchronization, where telemetry is published through MQTT and reflected in agent shadow states. The orchestration plane leverages the Kubernetes reconciliation loop for self-healing and elastic scaling, automatically restores failed agents, and enables dynamic service discovery. The architecture separates data and control planes to ensure low latency: high-rate telemetry flows directly to the time-series backend, while configuration and scaling operations are handled asynchronously through the Kubernetes API.

Architectural framework DTFarming is divided into three layers: the sensor-actuator, edge data center, and the User Application (Figure 4):

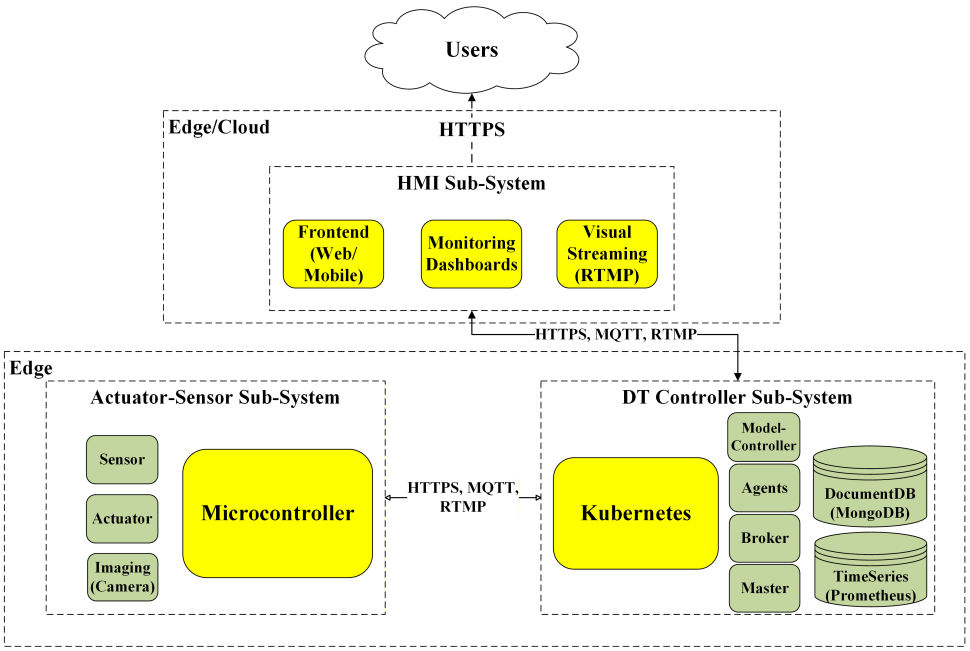
- Sensor-Actuator Layer: Consists of physical ESP32 microcontroller nodes interfacing with environmental sensors (DHT22, BH-1750, and MH-Z19B) and actuators. These nodes operate as "Device Twins," publishing state changes through MQTT.
- Edge Data Center: The core logic resides on a MicroK8s cluster hosted on a Raspberry Pi 5. This layer hosts the *Digital Twin Central (DTC)* services, which act as the orchestration hub.
- Service Layer: MongoDB handles data persistence for time-series storage, while Prometheus and Grafana provide real-time metrics database and visualization.

3. Results and Discussion

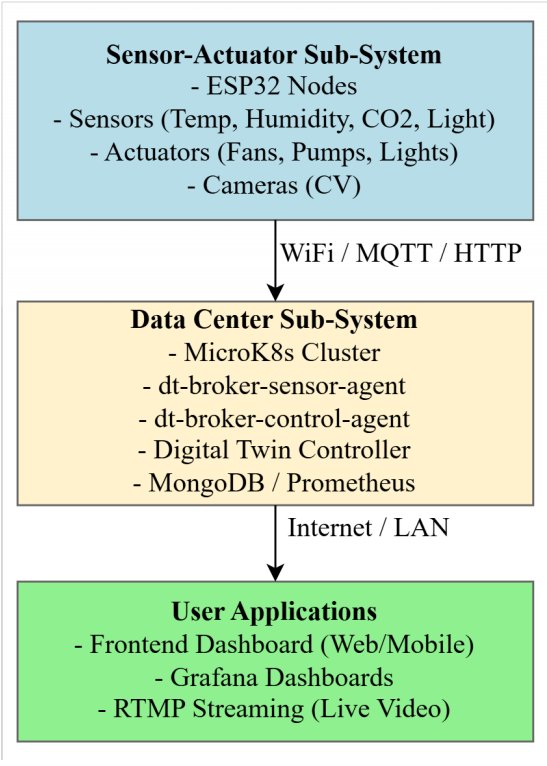
The implementation of physical hardware at the edge node uses a Raspberry Pi 5 (8GB RAM, Quad-core Cortex-A76), chosen for its capacity to handle container orchestration overhead. The sensor nodes employ ESP32-S3 modules connected through Wi-Fi. Communication follows an event-driven architecture.

MQTT is used for data exchange for low-latency telemetry and HTTP REST for state management and user commands. Table 2 shows a sample of the broker adapter REST API endpoint while Table 3 presents the standardized data formats for sensor measurements and control commands, ensuring consistency across DTFarming components. The Sensor Broker serves as an adapter middleware that normalizes heterogeneous sensor data into a standardized JSON format before transmission.

In this research, the sensor board uses an ESP32 module with three types of sensors in a customized board. Sensor board specifications, including the ESP32 board for the microcontroller device, three attached sensors devices: DHT22, BH-1750, and MH-Z19B, are shown in Figure 5(b) and Table 5.



(a)



(b)

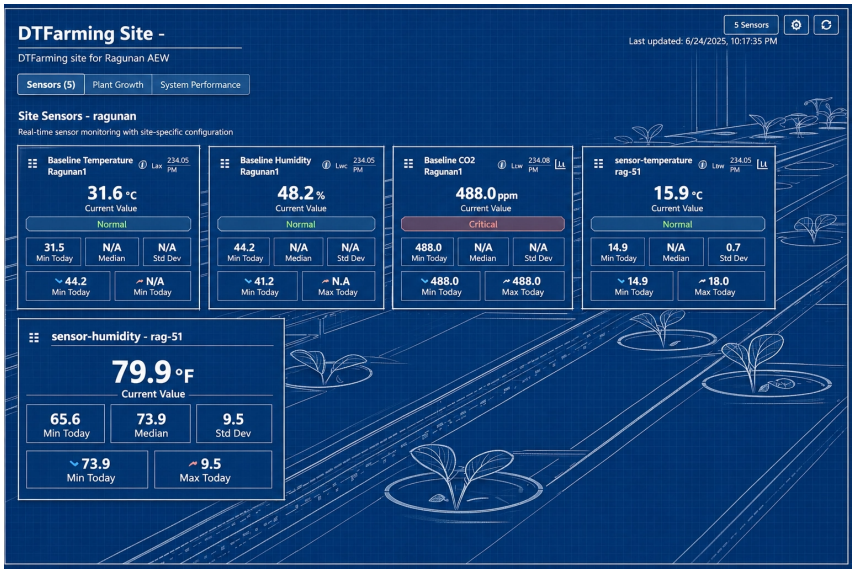
Figure 4 (a) DTFarming sub-system architecture and (b) block flow diagram

Table 2 Sample of Broker API Endpoint

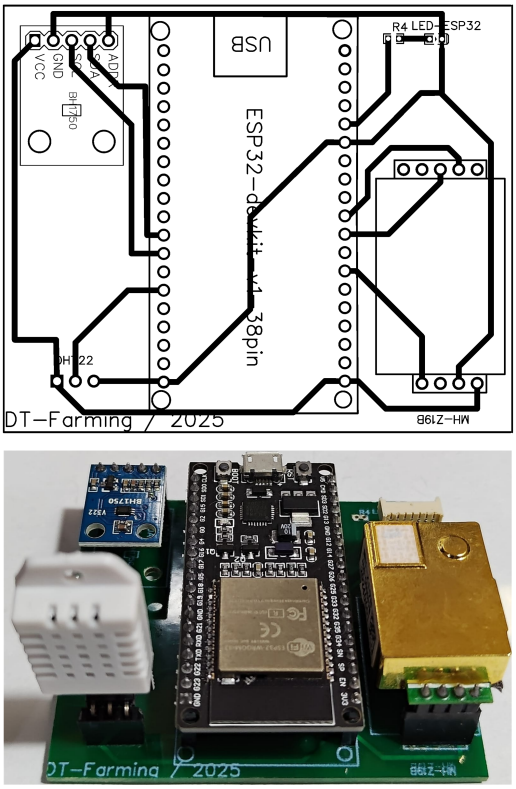
Endpoint	Method	Description
/api/sensor/data	POST	Accepts sensor readings in any format.
/api/sensor/data/latest	GET	Fetches the most recent sensor data.
/api/sensor/{sensor_id}	GET	Retrieves sensor-specific data.
/api/actuator/command	POST	Control command to an actuator.
/api/actuator/{actuator_id}/status	GET	Retrieves actuator status.

3.1 Physical Farming Environment

Figure 5(a) illustrates a sample of indoor farming systems deployed at Ragunan, a container farming facility in South Jakarta, Indonesia, for prototyping and testing. Further deployments and evaluations are in progress.



(a)



(b)

Figure 5 (a) DTFarming use case implementation (b) prototype sensor board

The system architecture employed a declarative Infrastructure as Code (IaC) paradigm, ensuring repeatable and consistent deployments (Kovács et al., 2024; Ristov et al., 2024), via HashiCorp Terraform to ensure environment consistency and to eliminate configuration drift. This approach automates the provisioning of the Kubernetes system, Helm charts for MongoDB/Mosquitto, and custom DTFarming microservice deployment. This declarative setup

reduces the manual deployment time from several hours to approximately 40 – 70 minutes.

Table 3 Sensor and control data format sample

Sensor data format	Control data format

<pre>{
"uuid": "5fda8ddd-b39f-42a7-8980-b14c1a2b827c",
"sensor_id": "temp-sensor-001",
"type": "temperature",
"unit": "Celsius",
"timestamp":"2025-02-25T12:34:56Z",
"value": 23.5,
"location": {
"farm_id": "farm-01",
"zone": "ragunan-2"
},
"metadata": {
"device_type": "DHT-22",
"board_mac": "00-1B-63-84-45-E6",
"description": ""
}
}</pre>
	<pre>{
	"uuid": "6299a1f1-1990-4b1b-8439-fc102c7d7d84",
	"actuator_id":"water-pump-001",
	"type": "pump",
	"command": "ON",
	"timestamp":"2025-02-25T12:35:10Z",
	"parameters": {
	"duration": 30,
	"flow_rate": 5
	},
	"status": "PENDING"
	}</pre>

Table 4 MicroK8s on a Raspberry Pi 5 (8GB RAM) on Ubuntu 24.04 LTS

Edge Server Hardware Specification	
Component	**Specification**
Board	Raspberry Pi 5
CPU	Quad-core ARM Cortex-A76, 2.4 GHz
RAM	8GB LPDDR4X
Storage	microSD 64GB
Networking	Ethernet, Wi-Fi (802.11ac)
Power Supply	5V 2-5A USB-C adapter
Software Specification	
Software	**Version**
OS	Ubuntu 24.04 LTS (64-bit)
Container Runtime	containerd (default MicroK8s)
Kubernetes	v1.30 (via MicroK8s)
MQTT Broker	Eclipse Mosquitto
Python	3.12
Flask	Latest
Paho MQTT	Latest

3.2 Evaluation and analysis of performance

The evaluation follows a controlled experimental design with three workload scenarios defined by the number of sensor boards and virtualized DT agents. To ensure repeatability, each experiment is executed on a clean system state after a full cluster restart. Sensor data are generated at fixed 30-second intervals, and Prometheus is used to collect metrics continuously.

Three experimental simulations were designed with distinct configurations and sensor densities to assess the scalability and reliability of the DTFarming system under different sensor loads:

1. **Baseline load (Simulation 1).** Configuration: Single board. Sensors: 4 total (DHT22 has two sensors, BH 1750 and MH-Z19B). This setup serves as a baseline for observing system behavior with minimal sensor load.

2. **Medium load (Simulation 2).** Configuration: Five boards. Sensors: 20 total. This scenario increases the sensor load fivefold, providing insights into how the system scales when managing moderate volumes of data and device connections.

3. **Peak load (Simulation 3).** Configuration: Twenty-five boards. Sensors: 100 total. In this larger-scale scenario, the capacity of the system to handle a high density of sensors and the associated data throughput was evaluated.

A comprehensive evaluation of Kubernetes scheduling and resource usage (CPU, memory, and network) was conducted. The Kubernetes scheduling rate—defined as the number of scheduled pod attempts per unit time—offers insight into how efficiently the Kubernetes scheduler (kube-scheduler) can respond to system load changes. The Kubernetes cluster was fully restarted before each experimental run to ensure repeatability. Sensor data were generated by ESP32-based sensor boards that transmit measurements at fixed intervals of 30 s.

Figure 6 shows that (i) with 4 sensors (Simulation 1), Kubernetes scheduling latency remained minimal, averaging 0.4 seconds per pod; (ii) with 20 sensors (Simulation 2), the scheduling time increased slightly to 0.5 seconds per pod, reflecting additional computational overhead; (iii) with 100 sensors (Simulation 3), Kubernetes scheduling latency increased to 2.1 s per pod, indicating a noticeable increase in orchestration overhead. The sharp increase in scheduling latency observed in Simulation 3 corresponds to a Scheduling Pressure Index (SPI) of 5.25 (Equation 3), indicating the onset of scheduler saturation rather than instability.

Although scheduling times increased, no pod restarts were needed, indicating stable performance under larger sensor loads. A full system restart was conducted before each new simulation to ensure consistent baseline conditions.

The CPU and memory consumption of core services (sensor agents, brokers, master service, and MongoDB) were analyzed under varying workloads, as shown in Figure 6 and Figure 7:

- Sensor agents exhibited CPU usage averaging 0.5–1.5 mCPU units per agent, with 17-18 MB of memory.

- Sensor brokers showed moderate CPU usages during low traffic, but it spiked 4-fold under high sensor loads but remained within acceptable limits. The sensor broker uses higher memory but remains stable in different scenarios.

- The master service required moderate memory allocation due to stateful operations and data persistence.

- MongoDB maintained a stable CPU footprint but showed a gradual increase in memory consumption as sensor data were stored. Database optimization with indexing could be an option for future optimization as larger data transactions are performed.

Under the peak load conditions in the third scenario, the Raspberry Pi node reached its capacity limit with 110 active pods, preventing further scaling. While CPU use remained low, memory usage was near saturation, indicating that memory constraints were the primary bottleneck in the system's scalability even if the node could reach more than 110 pods capacity.

As defined in Equation 1, the telemetry message arrival rate increases linearly with the number of sensors, given the fixed publish interval of 30 seconds. Therefore, the experimental configurations (4, 20, and 100 sensors) correspond to proportional increases in the offered load to the MQTT broker and DT services.

Under controlled deterministic arrivals (i.e $\Delta t = 30$s), the objective is to isolate the behavior of the orchestration layer from exogenous traffic burstiness. Eliminating stochastic input variability allows us to see if the system's delay (latency) spikes suddenly as memory fills up, purely

due to internal resource competition. Because the input is perfectly steady, the orchestrator's internal logic causes any sudden slowdown rather than random traffic spikes.

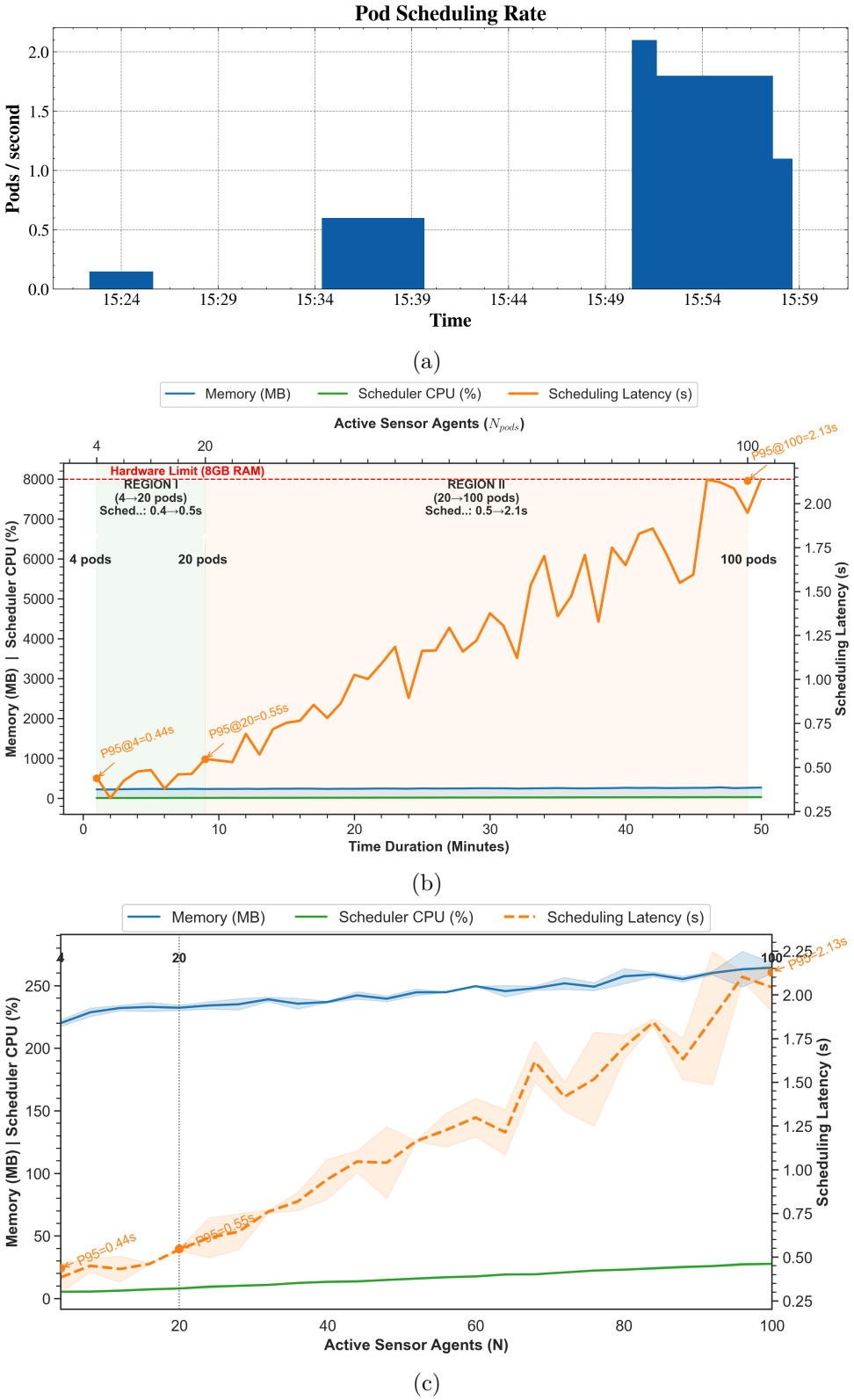


Figure 6 (a) Kubernetes Scheduling Rate and Restart (b) Agents CPU and (c) memory use of the agents

The MQTT–based communication exhibited stable and predictable data transmission patterns, Figure 8, with negligible packet loss. The throughput of the sensor broker demonstrated linear scalability with increasing sensor counts. The observed network throughput results (Table

8, Figure 8) confirm this linear growth pattern, with throughput scaling proportionally as the sensor count increases. This behavior validates the workload generation model and confirms the absence of artificial throttling or message loss in the nominal operating phase.

During testing, each sensor device transmitted readings at 30-second intervals, resulting in eight data transmissions per minute per board. With 80 active sensors, the network performance analysis over a one-hour period indicated sustained data transmission without packet loss, corresponding to several thousand sensor readings successfully transmitted.

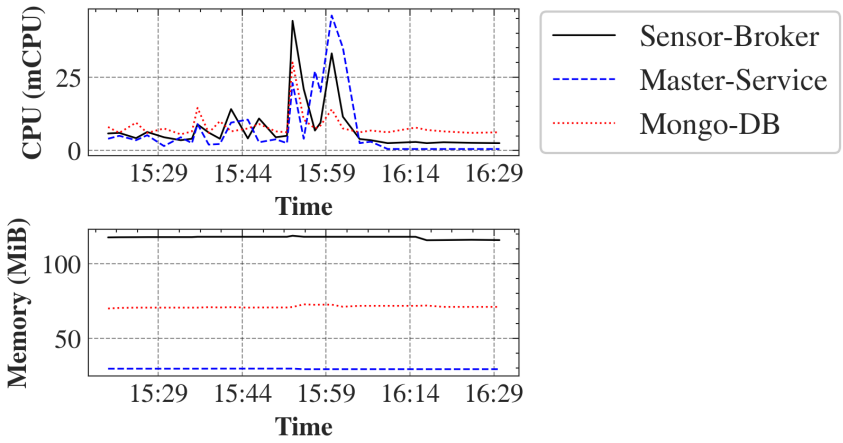


Figure 7 Database, Broker Service, Master Service resource load

Kubernetes scheduling performance was analyzed based on pod start time, scheduling rate, and system responsiveness under increasing sensor loads. The results are summarized in Table 6. During the experiments, the rate at which pods were scheduled increased proportionally with the number of sensor boards in use. Kubernetes can efficiently handle scheduling tasks across larger workloads, although higher sensor loads introduce mild delays in the time it takes to instantiate pods.

Kubernetes scheduling latency exhibits a nonlinear increase as the sensor density increases. While the average scheduling latency increases modestly from 0.4 to 0.5 s between 4 and 20 sensors, a sharp increase to 2.1 s is observed at 100 sensors (Table 6(a)). The Scheduling Pressure Index (SPI), defined in Equation 3, captures this behavior. Using the 4-sensor scenario as the baseline, the SPI increases to 1.25 at 20 sensors and reaches 5.25 at 100 sensors. Such an increase indicates the onset of scheduler saturation, as no pod failures or restarts were observed.

The Scheduling Pressure Index (SPI) in Table 6(b) increases from 1.00 at baseline to 5.25 at 100 sensors, demonstrating that scheduling latency amplifies more than fivefold under high-density deployment. This behavior confirms a memory-bound orchestration pressure-driven phase transition rather than CPU saturation, thereby validating the predictive scalability model.

Table 5 Sensor board specification

Sensor Board	Sensor device types
ESP32-S3 (WROOM-1) Dual-core Xtensa LX7 @ 240MHz, 512KB SRAM, 8MB Flash, 36 GPIO, 18 ADC, I^2C, UART, SPI, PWM	• DHT22: Sensor Temperature and Humidity • BH-1750: Sensor Light Intensity • MH-Z19B: Sensor CO_2

In this study, SPI captures the relative increase in scheduling delay, while the associated success rate, restart count, scheduler CPU usage, SD, and event-level CI provide supporting evidence that the observed increase reflects saturation pressure rather than deployment failure. The observed orchestration behavior is consistent with prior lightweight Kubernetes benchmarking studies, although this work emphasizes Digital Twin density scaling rather than platform com-

parison alone. Koziolek and Eskandani (2023) reported that lightweight Kubernetes distributions, such as K3s and K0s, generally exhibit lower control-plane overhead and faster orchestration responsiveness than more feature-complete configurations, while MicroShift demonstrated strong data-plane throughput under constrained edge conditions. Similarly, comparative studies involving MicroK8s, K3s, K0s, and Microshift indicate that under edge deployment conditions, resource utilization, scheduling responsiveness, and orchestration latency become increasingly sensitive to workload density and available memory capacity. The present findings are consistent with these observations, particularly the nonlinear increase in scheduling latency as the cumulative pod density approaches memory saturation. However, unlike prior benchmarking-oriented studies that primarily compare orchestration platforms using absolute throughput or latency metrics, this study focuses on the underlying orchestration dynamics governing Digital Twin agent container density growth. Therefore, the proposed SPI metric provides an additional orchestration-centric interpretation layer by quantifying scheduling latency amplification relative to a nominal DT deployment baseline.

Table 6 Summary of Kubernetes Scheduling Metrics

(a) Scheduling Summary

Metric	4 Sensors	20 Sensors	100 Sensors
Avg. Scheduling Time (s)	0.4	0.5	2.1
Scheduling Success Rate	100%	100%	99.8%
Pod Restart Count	0	0	0
Scheduler CPU Usage (%)	5%	8%	28%

(b) Summary of Confidence Interval and Scheduling Pressure Index

Scenario	Sensors (N_s)	Sched. Events (N_e)	Avg Sched. Latency (s)	SD(e) (s)	95% CI (s)	SPI
Baseline	4	12	0.40	0.030	[0.381, 0.419]	1.00
Moderate	20	28	0.50	0.045	[0.483, 0.517]	1.25
High	100	108	2.10	0.120	[2.077, 2.123]	5.25

These incremental scheduling slowdowns arose largely from Kubernetes distributing workloads and allocating resources even in a single worker node. Despite these short-lived delays, the platform maintained robust performance, with no service interruptions or restarts required. By incorporating more advanced scheduling strategies—such as the Horizontal Pod Autoscaler (HPA) or custom scheduling policies—the system could further mitigate instantiation delays and sustain reliable performance even as the sensor network grows.

The CPU consumption of core services (sensor agents, brokers, master service, and MongoDB) was analyzed under varying sensor loads. The key findings are presented in Table 7.

The analytical pod population model (Equation 9) predicts that the total number of pods increases linearly with the number of sensors because each independent DT agent represents one corresponding sensor. The experimental results confirm this assumption, with each additional sensor introducing a corresponding containerized agent without affecting the existing services.

As predicted by the per-agent resource cost model (Equation 10), CPU use per sensor agent remains approximately constant across all scenarios in Table 7(a). In contrast, memory consumption accumulates linearly with the sensor count, as shown in Table 7(b). This divergence explains why CPU use remains low, whereas memory becomes the dominant limiting factor.

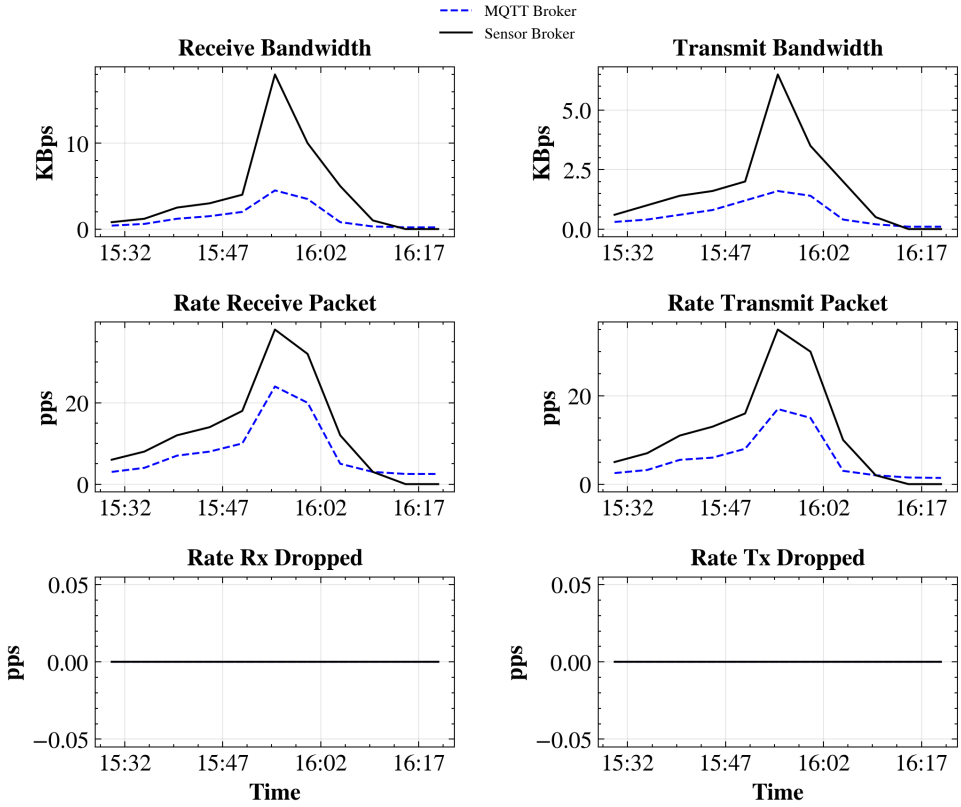


Figure 8 Network performance of the MQTT broker and sensor broker: (a) receive bandwidth, (b) transmit bandwidth, (c) received-packet rate, (d) transmitted-packet rate, (e) receive-drop rate, and (f) transmit-drop rate. The solid black line represents the sensor broker, and the dashed blue line represents the MQTT broker.

Table 7 (a) Summary of CPU Utilization by Service (b) Summary of Memory Utilization by Service

(a)

Service	4 Sensors	20 Sensors	100 Sensors
Sensor Agents	0.5–1.5 mCPU	0.5–1.5 mCPU	0.5–1.5 mCPU
Sensor Broker	5-8 mCPU	10-15 mCPU	15-45 mCPU
Master Service	5-8 mCPU	8-15 mCPU	10-45 mCPU
MongoDB	5-8 mCPU	8-12 mCPU	10-35 mCPU

(b)

Service	4 Sensors	20 Sensors	100 Sensors
Sensor Agents	17–18 MiB	17–18 MiB	17–18 MiB
Sensor Broker	100-110 MiB	110-120 MiB	110-130 MiB
MQTT Broker	2-3 MiB	5 MiB	15 MiB
Master Service	20-30 MiB	20-30 MiB	20-30 MiB
MongoDB	70 MiB	70-73 MiB	70-75 MiB

The observed scalability ceiling at ∼110 active pods directly aligns with the memory saturation threshold predicted by Equation 11, confirming that available memory, rather than processing capacity, primarily constrains system scalability.

The first 20 sets were running well when testing 25 sensor sets, but the last 5 sets were not successfully deployed. This showcases the ability of Kubernetes to isolate successful deployments from unsuccessful ones. As a result, 20 sets or 80 sensor agents are running without issues on a

single compute node with 4 GB or RAM.

The efficiency of MQTT–based communication was evaluated based on the message throughput, latency trends, and packet loss rates across various sensor loads. Table 8 presents the summary findings.

MQTT communication is highly scalable and resilient, maintaining near-zero packet loss up to 100 sensors. However, beyond this threshold, latency and queuing delays become significant bottlenecks. These findings identify adaptive MQTT message batching and edge-side preprocessing as potential future optimizations to reduce broker congestion and limit redundant sensor data transmissions.

Table 8 Summary of Network Performance Metrics

Metric	20 Sensors	50 Sensors	100 Sensors
Throughput (pps)	10.2	18.5	40+
Latency (ms)	35–50	40–55	80+
Packet Loss (%)	0.0%	0.0%	0.4%

The end-to-end DT responsiveness model (Equation 12) decomposes latency into network, processing, and orchestration components. The experimental results show that the network latency remains stable with near-zero packet loss up to 100 sensors (Table 8), indicating that the communication overhead is not the dominant contributor to responsiveness degradation.

Instead, the observed increase in scheduling latency under peak load conditions suggests that as the system enters the saturation phase, orchestration delay becomes the dominant term in Equation 12. This confirms the analytical expectation that orchestration overhead can outweigh communication delay at high deployment densities in edge-deployed containerized DT systems.

Reliability was also a key strength, as evidenced by the absence of pod restarts throughout the evaluations. The system consistently maintained real-time data flows and orchestrations even as the number of sensors increased, highlighting the robustness of both the sensor agents and the supporting microservices.

4. Future Research

This study uses a consistent, 30-second interval timed data flow to isolate how the system manages tasks; however, devices syncing clocks, mass reconnections, or sensors reacting to sudden events often face unpredictable spikes in real-world use. Extending the k8s workload model to stochastic arrivals (e.g., Poisson or self-similar burst processes) (Chamberlain et al., 2025; Kumar, 2025) and evaluating their impact on scheduling latency distribution and SPI sensitivity constitutes important future work. In addition, the reported confidence intervals quantify within-run variability across scheduling events, and independent multirun replication is required to assess between-run consistency. The present evaluation is also limited to a single node; therefore, to generalize the predictive model across larger distributed edge clusters, multinode scheduling dynamics, placement strategies, alternative orchestration platforms such as k0s, k3s, and MicroShift (Koziolek & Eskandani, 2023), and distributed resource fragmentation should be investigated in future studies. Beyond scalability, a further direction is the integration of IEC 61499 function block (FB)–based distributed control (IEC, 2022; Sonnleithner et al., 2021) through co-deployment strategies where IEC 61499 runtimes operate alongside containerized DT services, since scheduling pressure and memory saturation may directly influence the responsiveness of real-time distributed control functions, motivating dedicated scheduling strategies for DT agents and control FBs (Dai et al., 2023; Faqrizal et al., 2024). Embedding such coordination logic within IEC 61499 FB networks would allow formal reasoning about control safety and consistency in Digital Twin–driven farming systems (Garg et al., 2024; Shatrov & Vyatkin, 2020), and incorporating IEC 61499 execution semantics and control-loop timing constraints would further strengthen the foundations of DT–based distributed control (Verdouw et al., 2021).

5. Conclusion

This study investigates the scalability behavior of containerized Digital Twin deployments on resource-constrained edge nodes using an orchestration-centric performance model. The model relates sensor container density to workload arrival rate, pod population, per-agent resource cost, memory saturation, and scheduling pressure and is experimentally validated using deployments with 4, 20, and 100 sensor containers on a single MicroK8s edge node. The results provide empirical evidence that resource-driven phase transitions emerge intrinsically from Kubernetes orchestration dynamics as memory utilization approaches saturation. Telemetry throughput scales linearly with the sensor count with negligible packet loss, while the average CPU use remains below 25%, indicating low per-agent processing overhead. In contrast, cumulative memory consumption increases linearly, resulting in a predictable scalability ceiling at ∼110 active pods on an 8GB edge node. The analytically predicted memory-bound capacity (≈112 pods) deviates by less than 2% from the experimentally observed saturation threshold (∼110 pods). As this limit is approached, the Kubernetes scheduling latency increases nonlinearly from 0.4 to 2.1 s, corresponding to a 5.25 times increase in the SPI. The close alignment between analytical capacity prediction and observed saturation behavior supports the proposed scalability framework as a predictive tool for edge Digital Twin density planning. By isolating orchestration-layer effects under deterministic workload conditions, the study demonstrates that internal resource contention causes nonlinear scheduling growth.

Acknowledgements

This research was supported by the program of *Penelitian Pasca Sarjana-Penelitian Disertasi Doktor (PPS-PDD)* under grant contract number 015/C3/DT.05.00/PL/2025 (subcontract no. 343/IT1.B07.1/SPP-DRI/V/2025), provided by the Ministry of Higher Education, Science, and Technology, Indonesia.

Author Contributions

Conceptualization, AW, EJ, RL; methodology, AW, EJ, RL; software, RL; testing, RL, AW; performance analysis, RL, EJ; review, AW, EJ; All authors have read and agreed to the published version of the manuscript.

Conflict of Interest

The authors declare no conflicts of interest.

References

Alam, M., Rufino, J., Ferreira, J., Ahmed, S. H., Shah, N., & Chen, Y. (2018). Orchestration of microservices for iot using docker and edge computing. *IEEE Communications Magazine*, *56*(9), 118–123. https://doi.org/10.1109/MCOM.2018.1701233

Böhm, S., & Wirtz, G. (2021). Profiling lightweight container platforms: Microk8s and k3s in comparison to kubernetes. *CEUR Workshop Proceedings*, *2839*, 65–73.

Boursianis, A. D., Papadopoulou, M. S., Gotsis, A., Wan, S., Sarigiannidis, P., Nikolaidis, S., & Goudos, S. K. (2021). Smart irrigation system for precision agriculture - the arethou5a iot platform. *IEEE Sensors Journal*, *21*(16), 17539–17547. https://doi.org/10.1109/JSEN.2020.3033526

Burns, B., & Oppenheimer, D. (2016). Design patterns for container-based distributed systems. *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*. https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/burns

Centofanti, C., Tiberti, W., Marotta, A., Graziosi, F., & Cassioli, D. (2023). Latency-aware kubernetes scheduling for microservices orchestration at the edge. *2023 IEEE 9th Inter-*

national Conference on Network Softwarization (NetSoft), 426–431. https://doi.org/10.1109/NetSoft57336.2023.10175431

Chamberlain, J., Zheng, J., Zhu, Z., Liu, Z., & Starobinski, D. (2025). Exploiting kubernetes autoscaling for economic denial of sustainability. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, *9*(2). https://doi.org/10.1145/3727114

Dai, W., Zhang, Y., Kong, L., Christensen, J. H., & Huang, D. (2023). Design of industrial edge applications based on iec 61499 microservices and containers. *IEEE Transactions on Industrial Informatics*, *19*(7), 7925–7935. https://doi.org/10.1109/TII.2022.3214199

Fakeye, I., Maas, E., Harris, P., Oulaid, B., & Baker, C. (2024). Towards a framework for farm-scale digital twins. *Proceedings: MODELS 2024 - ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 486–491. https://doi.org/10.1145/3652620.3688264

Faqrizal, I., Salaün, G., & Falcone, Y. (2024). Adaptive industrial control systems via IEC 61499 and runtime enforcement. *ACM Transactions on Autonomous and Adaptive Systems*, *19*(4), 1–31. https://doi.org/10.1145/3691345

Garg, K., Usevitch, J., Breeden, J., Black, M., Agrawal, D., Parwana, H., & Panagou, D. (2024). Advances in the theory of control barrier functions: Addressing practical challenges in safe control synthesis for autonomous and robotic systems. *Annual Reviews in Control*, *57*, 100945. https://doi.org/10.1016/j.arcontrol.2024.100945

Gopalakrishnan, S., Waimin, J., Raghunathan, N., Bagchi, S., Shakouri, A., & Rahimi, R. (2021). Battery-less wireless chipless sensor tag for subsoil moisture monitoring. *IEEE Sensors Journal*, *21*(5), 6071–6082. https://doi.org/10.1109/JSEN.2020.3039363

Grieves, M. W. (2019). Virtually intelligent product systems: Digital and physical twins. In *Complex systems engineering: Theory and practice* (pp. 175–200). https://doi.org/10.2514/5.9781624105654.0175.0200

Gupta, P., Mokal, T. P., Shah, D. D., & Satyanarayana, K. V. V. (2018). Event-driven SOA-based IoT architecture. In *International conference on intelligent computing and applications* (pp. 247–258). Springer Singapore. https://doi.org/10.1007/978-981-10-5520-1_24

IEC. (2022). *IEC 61499-1:2012 function blocks - part 1: Architecture*. International Electrotechnical Commission (IEC).

Jeong, D.-Y., Baek, M.-S., Lim, T.-B., Kim, Y.-W., Kim, S.-H., Lee, Y.-T., Jung, W.-S., & Lee, I.-B. (2022). Digital twin: Technology evolution stages and implementation layers with technology elements. *IEEE Access*, *10*, 52609–52620. https://doi.org/10.1109/ACCESS.2022.3174220

Kjorveziroski, V., & Filiposka, S. (2022). Kubernetes distributions for the edge: Serverless performance evaluation. *Journal of Supercomputing*, *78*(11), 13728–13755. https://doi.org/10.1007/s11227-022-04430-6

Kovács, J., Ligetfalvi, B., & Lovas, R. (2024). Automated debugging mechanisms for orchestrated cloud infrastructures with active control and global evaluation. *IEEE Access*, *12*, 143193–143214. https://doi.org/10.1109/ACCESS.2024.3467228

Koziolek, H., & Eskandani, N. (2023). Lightweight kubernetes distributions: A performance comparison of microk8s, k3s, k0s, and microshift. *ICPE 2023 - Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, 17–29. https://doi.org/10.1145/3578244.3583737

Kristiani, E., Yang, C.-T., Huang, C.-Y., Wang, Y.-T., & Ko, P.-C. (2021). The implementation of a cloud-edge computing architecture using openstack and kubernetes for air quality monitoring application. *Mobile Networks and Applications*, *26*(3), 1070–1092. https://doi.org/10.1007/s11036-020-01620-5

KubeEdge. (2024). Kubeedge: Kubernetes native edge computing framework. https://github.com/kubeedge/kubeedge

Kumar, S. (2025). Oran-hautoscaling: A scalable and efficient resource optimization framework for open radio access networks with performance improvements. *Information*, *16*(4). https://doi.org/10.3390/info16040259

Li, H., Liu, X., & Zhao, W. (2023a). Research on lightweight microservice composition technology in cloud-edge device scenarios. *Sensors*, *23*(13). https://doi.org/10.3390/s23135939

Li, S., Li, X., & Yang, Z. (2023b). Research on temperature and light control system of vertical farm based on PLC. *Proceedings - 2023 International Conference on Networking, Informatics and Computing (ICNETIC 2023)*, 320–324. https://doi.org/10.1109/ICNETIC59568.2023.00073

Lubis, R., Widyotriatmo, A., Albert, & Joelianto, E. (2024). A microservices-driven digital twin model for precision farming in a controlled indoor environment. *2024 IEEE International Conference on Technology, Informatics, Management, Engineering and Environment (TIME-E)*, *5*, 180–185. https://doi.org/10.1109/TIME-E62724.2024.10919834

Madni, A. M., Madni, C. C., & Lucero, S. D. (2019). Leveraging digital twin technology in model-based systems engineering. *Systems*, *7*(1). https://doi.org/10.3390/systems7010007

Nasirahmadi, A., & Hensel, O. (2022). Toward the next generation of digitalization in agriculture based on digital twin paradigm. *Sensors*, *22*(2). https://doi.org/10.3390/s22020498

Pan, Z., Hur, B., Myles, K., & Adelman, Z. (2022). Development of raspberry Pi 4 B and 3 B+ micro-kubernetes cluster and iot system for mosquito research applications. *Computation*, *10*(12). https://doi.org/10.3390/computation10120221

Pylianidis, C., Osinga, S., & Athanasiadis, I. N. (2021). Introducing digital twins to agriculture. *Computers and Electronics in Agriculture*, *184*. https://doi.org/10.1016/j.compag.2020.105942

Ristov, S., Brandacher, S., Hautz, M., Felderer, M., & Breu, R. (2024). CODE: code once, deploy everywhere serverless functions in federated FaaS. *Future Generation Computer Systems*, *160*, 442–456. https://doi.org/10.1016/j.future.2024.06.017

Senjab, K., Abbas, S., Ahmed, N., & Khan, A. U. R. (2023). A survey of kubernetes scheduling algorithms. *Journal of Cloud Computing*, *12*(1), 87. https://doi.org/10.1186/s13677-023-00471-1

Shatrov, V., & Vyatkin, V. (2020). Formal verification of iec 61499 enhanced with timed events. *Doctoral Conference on Computing, Electrical and Industrial Systems*, 168–178. https://doi.org/10.1007/978-3-030-45124-0_16

Sonnleithner, L., Wiesmayr, B., Ashiwal, P., & Zoitl, A. (2021). IEC 61499 distributed design patterns. *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–8. https://doi.org/10.1109/ETFA45728.2021.9613569

Tao, F., Zhang, H., Liu, A., & Nee, A. Y. C. (2019). Digital twin in industry: State-of-the-art. *IEEE Transactions on Industrial Informatics*, *15*(4), 2405–2415. https://doi.org/10.1109/TII.2018.2873186

Turnip, A., Pebriansyah, F. R., Simarmata, T., Sihombing, P., & Joelianto, E. (2023). Design of smart farming communication and web interface using MQTT and node.js. *Open Agriculture*, *8*(1). https://doi.org/10.1515/opag-2022-0159

Usman, M., Ferlin, S., & Brunstrom, A. (2024). Performance analysis of lightweight container orchestration platforms for edge-based IoT applications. *Proceedings - 2024 IEEE/ACM Symposium on Edge Computing (SEC 2024)*, 321–332. https://doi.org/10.1109/SEC62691.2024.00032

Verdouw, C., Tekinerdogan, B., Beulens, A., & Wolfert, S. (2021). Digital twins in smart farming. *Agricultural Systems*, *189*. https://doi.org/10.1016/j.agsy.2020.103046