Chapter 3

Secure Task Containerization with Deadline Constraint

Industrial IoT (IIoT) revolutionizes digital manufacturing by incorporating a diverse array of devices, simulators, and tools, many of which are equipped with multiple sensors. These sensors play a crucial role in providing ample data for the coordination and monitoring of industrial systems. Traditionally, IIoT necessitates dedicated supporting devices to ensure that specific applications can execute tasks within the maximum allowable response time. However, the reliance on dedicated devices inherently inflates system costs. Task containerization offers an innovative solution to this issue. It is a process that harnesses the available resources of host machines, efficiently meeting the varying demands of different applications without the need for dedicated IIoT devices, thus mitigating the increased costs that arise from their use. It involves encapsulating software tasks within isolated containers which will combine the benefits of task isolation, security, and meeting specific deadlines. This approach is crucial in industrial and IoT applications where both data security and timely task execution are paramount. It ensures that tasks are executed securely within specified time limits, making it a fundamental element in various critical systems. This chapter introduces a novel approach

dedicated to securely processing IIoT tasks within the confines of the allowable response time. The approach utilizes the principles of game theory to estimate the fractions of tasks suitable for containerization on different machines. In addition to optimizing resource utilization, reducing the need for dedicated devices, and thereby lowering costs in Industry 4.0, the chapter also sheds light on the security vulnerabilities associated with containerization and presents methods to address these concerns. Ultimately, the estimated fractions for each machine aim to maximize the system's utility, and this chapter substantiates its effectiveness through experimental results that validate the performance of the proposed approach.

3.1 Introduction

The advent of IIoT has revolutionized digital manufacturing by incorporating a plethora of devices, simulators, and tools equipped with multiple sensors. These sensors furnish an abundance of data, facilitating the coordination and monitoring of industrial systems. However, IIoT often necessitates dedicated devices tailored to specific applications to execute tasks within the maximum allowable response time. The requirement for such dedicated devices results in increased system costs. Embracing the concept of task containerization is a fundamental shift in this landscape by optimizing the utilization of host machine resources to meet the diverse demands of various applications. Task containerization involves encapsulating tasks within isolated containers, ensuring efficient execution and resource utilization. This approach mitigates the need for purchasing dedicated IIoT devices every time a new application is introduced, thereby reducing costs. This chapter introduces a novel approach to securely process IIoT tasks within the confines of acceptable response times.

In the context of Industry 4.0, the incorporation of the Internet of Things is extended and referred to as the IIoT [**?**, **?**]. Industry 4.0's evolution often mandates the acquisition of dedicated devices to accommodate new applications. Consequently, the

integration of new features demands additional dedicated supporting devices and their interconnection [**?**]. This growing demand for dedicated devices to run new IIoT applications not only escalates costs but also introduces security vulnerabilities to Industry 4.0 [**?**]. Task containerization emerges as a transformative approach to address these challenges.

Containerization serves as a form of operating system virtualization that allows multiple containers to share the operating system of a host machine. This sharing optimizes resource utilization, thereby alleviating cost inefficiencies in Industry 4.0. Containers are self-contained units that help reduce the software overhead imposed by virtual machines. They are particularly advantageous in IIoT applications designed for long lifetimes. Containerization facilitates the replacement of hardware in an IIoT application after a predefined interval while supporting the seamless transition of applications to new hardware without dependencies on the local environment. Docker swarm, or simply swarm, is an example of a native clustering engine [**?**]. Fig. 3.1 illustrates a swarm where a Master Machine (MM) is managing the entire cluster and handles the available resources at Worker Machine (WM).

Despite various advantages of the swarm, the rapid increase in the number of containers on the different host machines increases the chances of security compromise. Thus, container security has drawn the attention of researchers. The attackers can use existing swarm containers or can create new containers. The attackers can enter and infect the containers and MM in such cases. Later, the attacker can access and modify the existing files of the swarm. For example, some recent attacks scanned for Docker servers are Doki, Ngrok, Kinsing, XORDDOS, AESDDOS, Team TNT, Xanthe, *etc.* Recently, various tools and antiviruses have been developed to identify vulnerabilities by scanning the containers [**?**]. Examples of such tools are Anchore, Clair, Dagda, OpenSCAP, and Sysdig Falco. A WM in the swarm generates the revenue when it successfully executes the assigned task by MM. However, WM requires some financial expenditure

in purchasing security tools and task processing. Moreover, block-chain [**?**, **?**] and data integrity checking [**?**] are also utilized to provide secure task executions in IIoT. Wang *et al.* in [**?**] proposed a novel certificate-less signature, utilizing block-chains. It improved the security weakness of the certificate-less signature.

In this chapter, we focus on secure processing of a given IIoT task within Maximum Allowable Response (MAR) time on the containers of WMs. We assume that a system consists of a MM and multiple WMs, where each WM has multiple containers to process the task. We propose an approach to address the following problem: *how the master machine containerizes the sub-tasks on the worker machines such that the task is processed in the MAR time with desired security level?* We refer to this problem as (α, β)-Task Containerization problem, or simply **(α, β)-TC** problem, where α and β are desired security level and MAP time to process the task, respectively. The introduction of task containerization methods enhances the efficiency of task processing and ensures compliance with security and time constraints.

In summary, this chapter responds to critical challenges posed by IIoT and Industry 4.0, particularly in terms of cost-efficiency, security, and task optimization. By introducing a game theory-based approach to task containerization and providing experimental results, the chapter offers innovative solutions for secure and efficient IIoT task processing within specified time constraints. This research contributes significantly to advancing the field of IIoT and Industry 4.0 by addressing these pivotal concerns. Task containerization stands out as a key enabler in addressing the core challenges posed by IIoT and Industry 4.0.

3.1.1 Major Contribution

To solve **(α, β)-TC** problem, we build a game theory model for securely containerizing the task on the WMs while maximizing the utility of MM. We use the Stackelberg game, where WMs and MM are the followers and leader, respectively. To maximize

its revenue, the leader determines the fraction of the task containerizing on the WMs. WMs charge some cost to execute the given fractions of the task, as per the price they paid for desired security. The outcome of the game model is the optimal strategy of MM, *i.e.*, the fractions of the task to be containerized on the WMs. Next, we prove the existence and uniqueness of the equilibrium for the follower and leader games. Further, we present a distributed algorithm, where WMs are participated based on the available resources. Finally, we validate the analysis and demonstrate the significance of the number of LNs, LGs, services, transmission rate, and CRs in estimating time duration to use the allocated SFs. We use network simulator, which provides the flexibility to study a relatively large number of different LoRaWAN scenarios.

The rest of the chapter is organized as follows: In the next section, we summarize the overview of existing literature on IIoT tasks container security mechanism. The next section introduces the related work. Section 3.3 presents the system model and preliminaries used in this work. Further, Section 3.4 presents the solution of (α, β)**-TC** problem. We present the experimental evaluation of the proposed work in Section 3.5. The chapter ends with the conclusion in Section 3.6.

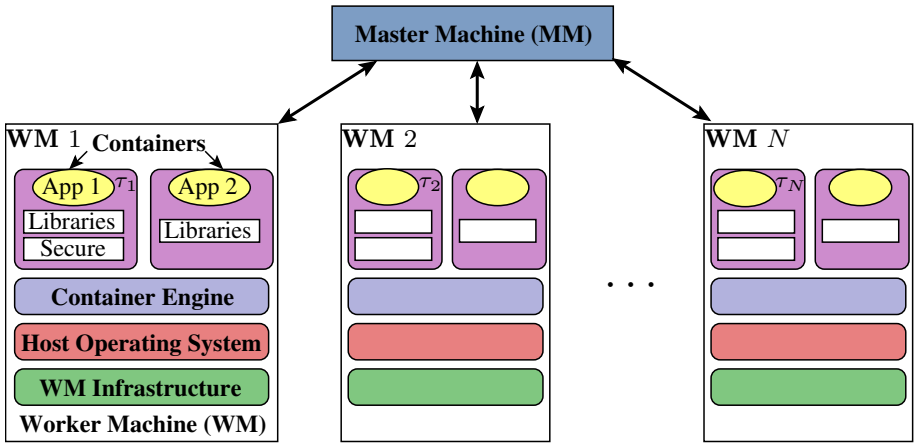


Figure 3.1: Illustration of swarm network with master and worker machines, where τ_n is the fraction of task executed on WN n.

3.2 Background and Motivation

We can divide the existing literature on the prevention of attack within containerized into two parts, *i.e.,* host machine-dependent and independent.

3.2.1 Host Machine Dependent

The security policies on the host machine can be propagated to the containers to enhance their security. The aspect was covered by the authors in [**?**]. Initially, the static set of security rules is assigned to the containers. Later, these rules are dynamically updated using additional rules to achieve high-order security. Similarly, in [**?**], the authors proposed a security namespace, which allows the container to explicitly implement its security policies with the existing policies.

3.2.2 Host Machine Independent

The security policies on the containers are independent of the host machines. Thus, it is more convenient but sometimes cost-efficient. The authors in [**?**] proposed a containerization Edge computing paradigm, namely CUTE. It facilitates low-latency services in vehicular networks. The security aspect of the CUTE is handled by using the blockchaining technique. Next, the authors in [**?**] introduced the concept of hardware like a virtualized trusted platform module for containerization. It improves the security of the containers. Finally, the authors [**?**] introduced a mechanism for the unification of security threats in virtualization or containerization. Table 3.1 illustrates the summary of existing work on container security.

3.2.3 Motivation

This work is motivated by the following limitations as noted in the existing literature. The existing literature on task containerization involves heterogeneous containers [**?**, **?**,

Table 3.1: Summary of the existing literature on container security.

Reference	Year	Technique	Mechanism	Machine dependency	Cost optimization
Lin *et al.* [?]	2020	Application classification	Detection	✗	✗
Zheng *et al.* [?]	2021	Inheritance graph	Detection	✗	✗
Cui *et al.* [?]	2021	Block chaining	Prevention	✗	✗
Andreou *et al.* [?]	2018	Auto-updating security policy	Prevention	✗	✗
Sun *et al.* [?]	2018	Security Namespace	Prevention	✓	✗
Hosseinzadeh *et al.* [?]	2016	Trusted platform module	Prevention	✓	✗
Zerouali *et al.* [?]	2019	Relating vulnerable bugs	Detection	✗	✗
Catuogno *et al.* [?]	2016	Secure virtualization	Prevention	✗	✗

?,?,?] and facilitates optimal task allocation on containers [?,?,?,?]. However, none-of-these these works covered the aspect of container security on the host machine. Next, to enhance the system performance and accelerate the task execution, parallel processing on all the containers is mandatory. Thus, task partitioning and simultaneous execution on multiple containers would be beneficial. However, the task partitioning aspect in containerization is unexplored in the existing literature [?, ?, ?, ?, ?, ?]. Additionally, increasing overall system utility is uncovered in the prior studies, which proves beneficial for the wider acceptability of containerization. Further, incorporation of the security mechanism in the existing work [?,?,?,?,?,?,?,?,?] on containerization do not emphasize on cost optimization.

3.3 System Model and Preliminaries

This work uses a hierarchical Docker Swarm network topology which consists of one MM and N numbers of WMs. MM assigns the task to the WMs (*e.g.*, laptop, desktop) in such a way that the task is completed successfully, incurs the minimum cost and

achieves desired security. Some attackers are also present in the network to disseminate malware to many users and their vulnerable machines. WMs purchase antivirus software to protect data from the attacker based on the probability of attack. Let $\mathcal{N}$ is the set of N number of WMs *i.e.,* $\mathcal{N} = \{1, \ldots, n, \ldots, N\}$ and one MM in the network which is denoted by M. Let WM n create Z_n number of containers to perform the assigned task by MM. The MM has a complete task $\mathcal{D}$ to be executed on WMs.

- **Data Transmission Delay:** The data transmission time between MM M and WM n depends on the transmission rate of the communication channel between them. Let B_n, h_n, and z_n represent the bandwidth, channel coefficient, and path loss frequency between MM and WM n, respectively. The data transmission rate $\mathcal{R}_n$ from a WM n to MM M can be calculated using Shannon channel capacity [**?**] $\mathcal{R}_n = B_n \log_2\left(1 + \frac{z_n h_n^2}{\sigma^2}\right)$. Given $\mathcal{R}_n$ and fraction of data τ_n assigned at WM n from $\mathcal{D}$, the required time for transmission from WM n to M is: $T_n^t = \frac{\tau_n \mathcal{D}}{\mathcal{R}_n} = \frac{\tau_n \mathcal{D}}{B_n \log_2\left(1+\frac{z_n h_n^2}{\sigma^2}\right)}$.
- **Data computation delay:** Let $L_{n,z}$ denotes the computation capability of a container z of WM $n \in \mathcal{N}$ in terms of CPU cycles runs per unit time. The computation time T_n^c for the execution of task $\zeta_{n,z} = \pi_{n,z}\tau_n\mathcal{D}$ bits on container z at WM n is $T_n^c = \sum_{z=1}^{Z_n} \frac{\zeta_{n,z} \times \mu_{n,z}}{L_{n,z}}$, where $\mu_{n,z}$ and $\pi_{n,z}$ denote CPU cycles required for processing one bit and the fraction of task assigned at container z, respectively.
- **Result Transmission Delay:** Transmitting result of size d bits from WM $n \in \mathcal{N}$ to MM M incurs result transmission delay, as: $T_n^r = \frac{d}{\mathcal{R}_n} = \frac{d}{B_n \log_2\left(1+\frac{z_n h_n^2}{\sigma^2}\right)}$.

The total delay $\mathbb{T}_n = T_n^t + T_n^c + T_n^r$ is the sum of data transmission, computation, and result transmission delays.

3.4 Solution of (α, β)-TC Problem

This section first designs a follower game among $\mathcal{N}$ WMs to estimate the optimal price γ_n charges from w_0.

3.4.1 Estimation of Optimal Price for Containerizing the Task

This section designs a follower game with WMs as players to estimate the optimal price charges by w_n $(1 \leq n \leq N)$ from w_0 to containerize the task. Such price maximizes the utility of w_n and satisfies the container security threshold α. The utility function of w_n consists of the following terms:

3.4.1.1 Price Gain from MM

The w_n demands a price (denoted as γ_n) to containerize the task assigned by w_0. Let Z_n be the number of required containers for τ_n, $\gamma_n = \sum_{z=1}^{Z_n} \gamma_{n,z}$. As per the oligopoly market, w_n reduces $\bar{\gamma}_{n,z}$ price from their demanded price if any other w_m increases price more than the threshold value, where $1 \leq \{m, n\} \leq N$ and $m \neq n$. We can define $\bar{\gamma}_{n,z}$ as an indicator function $\mathbb{1}_{\{\gamma_{n,z} > \gamma^{th}\}}$. The indicator function gives value 1 when $\gamma_{n,z} > \gamma^{th}$ and 0 otherwise. The price function given by w_0 to w_n per unit data, with the pricing coefficient δ, is defined as:

$$\mathcal{P}_n(\boldsymbol{\gamma}) = \sum_{z=1}^{Z_n} (\gamma_{n,z} - \bar{\gamma}_{n,z}). \tag{3.1}$$

Let τ_n be the fraction of task $\mathcal{D}$ assigned to w_n from w_0. Further, w_n assigns $\zeta_{n,z} = \pi_{n,z}\tau_n\mathcal{D}$ fraction of task to the container z. The total price gain of w_n depends on the probability of container security $1 - \alpha_n$ *i.e.*, the price paid by w_0 to w_n only for those tasks which are securely containerized. α_n is the probability of successful

attack. On increasing the price $\gamma_{n,z}$, some amount $(Q_{n,z})$ from the price is reduced due to antivirus software quality improvement. Therefore, the total price gain of w_n is the product of price function, assigned data, probability of container security, and quality of data, *i.e.*,

$$L_g(\boldsymbol{\gamma}_n) = \sum_{z=1}^{Z_n} \mathcal{P}_n(\boldsymbol{\gamma}) \zeta_{n,z} (1 - \alpha_n) Q_{n,z}, \tag{3.2}$$

where, $Q_{n,z} = c_q \log(1 - \frac{\gamma_{n,z}}{\gamma_{max}})$, c_q is the price coefficient to give the same magnitude as the quality of data and γ_{max} is the maximum allowable price.

3.4.1.2 Task Execution Cost

Let Z_n be the number of containers at w_n required to execute $\tau_n D$ portion of the task assigned by w_0. $\pi_{n,z}$ denotes the fraction of task given to a container z at w_n, where $1 \leq z \leq Z_n$. The computational capacity per instruction $\mu_{n,z}$ is decided by the frequency of Central Processing Units (CPUs) equipped at w_n and the quality of data $Q_{n,z}$. If the cost per CPU cycle is denoted by c_l and modelled as a quadratic function, the total execution cost at w_n is given as:

$$L_e(\boldsymbol{\gamma}_n) = c_l^2 \sum_{z=1}^{Z_n} Q_{n,z} \zeta_{n,z} \mu_{n,z}. \tag{3.3}$$

3.4.1.3 Cost Paid for Security Mechanism

w_n purchases a security mechanism to provide the security of the task. It consists of two terms; security mechanism purchasing cost $C_{n,s}$ and additional cost $C_{n,a}$ on increasing the size of data. Let l_n be the limit at the data which is scanned by the purchased security mechanism at the cost of $C_{n,s}$. Additional cost includes the quadratic increase in cost as the data size increases. The cost paid for the security mechanism by w_n is

given as:

$$\begin{aligned} L_a(\boldsymbol{\gamma}_n) =& C_{n,s} + C_{n,a} \\ =& C_{n,s} + (l_n C_{n,s}^2 + l_n C_{n,s}^3 + \cdots + l_n C_{n,s}^{\frac{\tau_n \mathcal{D}}{l_n}}), \\ =& C_{n,s} + l_n C_{n,s}^2 \left(\frac{C_{n,s}^{\frac{\tau_n \mathcal{D}}{l_n}-1} - 1}{C_{n,s} - 1} \right). \end{aligned} \tag{3.4}$$

The utility function of w_n is sum of price it receives by w_0, and cost paid for its resources and security mechanism, *i.e.*,

$$U_n^F(\boldsymbol{\gamma}_n, \tau_n) = L_g(\boldsymbol{\gamma}_n) - L_e(\boldsymbol{\gamma}_n) - L_a(\boldsymbol{\gamma}_n). \tag{3.5}$$

$$\begin{aligned} \textbf{Problem 1:} \max_{\boldsymbol{\gamma}_n} \quad & U_n^F(\boldsymbol{\gamma}_n, \tau_n) && \text{(3.6a)} \\ \text{s.t.} \quad & C_{n,s} + C_{n,a} < \sum_{z=1}^{Z_n} \gamma_{n,z} < \gamma_{max}, && \text{(3.6b)} \\ & (1 - \alpha_n) \geq \alpha, && \text{(3.6c)} \end{aligned}$$

where $1 \leq n \leq N$ and γ_{max} denote a constraint that imposes a limit on the maximum allowable price for the service.

Theorem 3.1 *Let γ_n be the strategy of w_n WM for containerizing a task of w_0 MM, where $1 \leq n \leq N$. The best response $\gamma_n^\star$ of w_n is given by:*

$$\gamma_n{}^{\star} = \sum_{z=1}^{Z_n} \gamma_{n,z}^{\star} = \sum_{z=1}^{Z_n} \frac{ac_l^2 \zeta_{n,z} \mu_{n,z} - \lambda_{n,2}}{a\zeta_{n,z}(1-\alpha_n)(2-\delta-\delta)\gamma_{-n}},$$

$$\textit{where,} \quad \lambda_{n,2} = \frac{1}{\sum_{z=1}^{Z_n} \frac{1}{S_n}} \left(\sum_{z=1}^{Z_n} \frac{ac_l^2 \zeta_{n,z} \mu_{n,z}}{S_n} - \gamma_{max} \right), \textit{ and} \qquad \gamma_{-n} = \sum_{i \neq n}^{N} \sum_{z=1}^{Z_i} \gamma_{i,z}.$$

Proof: Using the Taylor series expansion of $\log\left(1 - \frac{\gamma_{n,z}}{\gamma_{max}}\right)$ and neglecting the higher degree of $\frac{\gamma_{n,z}}{\gamma_{max}}$ due to decimal value of the fraction, the quality of task for w_n on container z can be rewritten as: $Q_{n,z} \approx \bar{Q}_{n,z} = -\frac{\gamma_{n,z}}{\gamma_{max}} - \frac{\gamma_{n,z}^2}{2\gamma_{max}^2} = -a\gamma_{n,z} - b\gamma_{n,z}^2$. Consider the utility function of w_n from Eq. 3.5. Using Lagrangian multipliers $\lambda_{n,1}$ and $\lambda_{n,2}$ for constraints defined in Eq. 3.6 for w_n, which satisfies the security constraint α, the Lagrangian of Eq. 3.6 can be represented as:

$$\begin{aligned} &\mathcal{L}_n^F(\boldsymbol{\gamma}_n, \tau_n, \lambda_{n,1}, \lambda_{n,2}) = \\ &\quad \sum_{z=1}^{Z_n} \mathcal{P}_n(\boldsymbol{\gamma}) \zeta_{n,z}(1-\alpha_n) \bar{Q}_{n,z} - c_l^2 \sum_{z=1}^{Z_n} \bar{Q}_{n,z} \zeta_{n,z} \mu_{n,z} - C_{n,s} \\ &\quad - C_{n,a} - \lambda_{n,1}(\gamma_{n,z} - \gamma_{max}) + \lambda_{n,2}(\gamma_{n,z} - C_{n,s} - C_{n,a}), \end{aligned} \tag{3.7}$$

where, the Lagrangian multipliers $\lambda_{n,1}$ and $\lambda_{n,2}$ are the non-negative dual variable. The dual function is $q(\lambda_{n,1}, \lambda_{n,2}) = \max_{\boldsymbol{\gamma}_n} \mathcal{L}_n^F(\gamma_n, \tau_n, \lambda_{n,1}, \lambda_{n,2})$. The Lagrange dual problem is then given by $\min_{\lambda_{n,1} \geq 0, \lambda_{n,2} \geq 0} q(\lambda_{n,1}, \lambda_{n,2})$. The duality gap is zero for the convex problem addressed; thus, solving its dual problem is equivalent to solving the original problem. Therefore, the optimal solutions needs to satisfy the following Karush-Kuhn-Tucker (KKT) conditions: $\lambda_{n,1}(\gamma_{n,z} - \gamma_{max}) = 0$, $\lambda_{n,2}(\gamma_{n,z} + C_{n,s} + C_{n,a}) = 0$, and $\lambda_{n,1} > 0, \lambda_{n,2}\gamma_{n,z}, \lambda_{n,2}, \lambda_{n,2}\gamma_{n,z} \geq 0$.

The first and second order derivatives of $\mathcal{L}_n^F(\boldsymbol{\gamma}_n, \tau_n, \lambda_{n,1}, \lambda_{n,2})$ *w.r.t.* $\gamma_{n,z}$ are $\zeta_{n,z}(1-$

$\alpha_n)(\mathcal{P}(\boldsymbol{\gamma_n})\hat{Q}_{n,z} + \bar{Q}_{n,z}(1-\delta)) - c_l^2\zeta_{n,z}\mu_{n,z}\hat{Q}_{n,z} + \lambda_{n,1} - \lambda_{n,2}$ and $-2\zeta_{n,z}(1-\alpha_n)(b\mathcal{P}(\boldsymbol{\gamma}_n) - \hat{Q}_{n,z}(1-\delta)) + 2bc_l^2\zeta_{n,z}\mu_{n,z}$, respectively, where $\hat{Q}_{n,z} = -a - 2b\gamma_{n,z}$. Since, $\gamma_{n,z}$ is the base price which is much higher than the term $2bc_l^2\zeta_{n,z}\mu_{n,z}$, we can conclude that the second order derivative of $\mathcal{L}_n^F(\boldsymbol{\gamma}_n, \tau_n, \lambda_{n,1}, \lambda_{n,2})$ is negative, the payoff function in Eq. 3.6 is concave and continuous; therefore, the follower level game has at least one SE. Let A and C are the coefficient matrix, Eq. 3.6 can be rewritten as: $A \times \gamma = C$, where, $A_{n,n} = 2a\zeta_{n,z}(1-\alpha_n)(1-\delta)$, $A_{n,l} = -a\zeta_{n,z}(1-\alpha_n)\gamma_{l,z}$, and $c_n = c_l^2\zeta_{n,z}\mu_{n,z}$. From the strictly diagonal dominant theorem [**?**], the matrix A is non-singular and the inverse of the matrix A is possible. Best response strategies of $\{w_1, w_2, \cdots, w_N\}$ WMs as the price $\gamma = [\gamma_1, \cdots, \gamma_n, \cdots, \gamma_N]$ to provide services to w_0 MM can be calculated as $A^{-1}C$, which is defined as:

$$\gamma_{n,z}{}^{\star} = \frac{ac_l^2\zeta_{n,z}\mu_{n,z} + \lambda_{n,1} - \lambda_{n,2}}{a\zeta_{n,z}(1-\alpha_n)(2-\delta-\delta\sum_{i\neq n}\sum_{z=1}^{Z_i}\gamma_{i,z})}. \tag{3.8}$$

Substituting $\gamma_{n,z}$ in the constraint of Eq. 3.7, we get:

$$\lambda_{n,2} = \frac{1}{\sum_{z=1}^{Z_n}\frac{1}{S_n}}\left(\sum_{z=1}^{Z_n}\frac{ac_l^2\zeta_{n,z}\mu_{n,z}}{S_n} - \gamma_{max}\right), \tag{3.9}$$

where, $S_n = a\zeta_{n,z}(1-\alpha_n)(2-\delta-\delta\sum_{i\neq n}\sum_{z=1}^{Z_i}\gamma_{i,z})$. $\lambda_{n,1} = 0$ and $\lambda_{n,2}$ from Eq. 3.9 into Eq. 3.8 and hence proved. □

Lemma 3.1 *The follower game has a unique NE if it satisfies the following three conditions.*

Positivity: $\forall\gamma_n \geq \gamma'_n$, *and* $n \in \mathcal{N}$*:*

$$U_n^F(\gamma_n, \gamma_{-n}, \tau_n) - U_n^F(\gamma'_n, \gamma_{-n}, \tau_n) \geq 0.$$

Monotonicity: $\forall\gamma_n \geq \gamma'_n$, *and* $n \in \mathcal{N}$*:*

$$|\Delta U_n^F(\gamma_n, \gamma'_n, \gamma_{-n}, \tau_n) - \Delta U_n^F(\gamma_n, \gamma'_n, \gamma'_{-n}, \tau_n)| \leq 0.$$

Scalability: *There is a* $\mu \geq 1$ *such that:*

$$\kappa(U_n^F(\gamma_n, \gamma_{-n}, \tau_n)) - U_n^F(\kappa(\gamma_n, \gamma_{-n}, \tau_n)) \geq 0.$$

Proof: It is seen that the second derivative of utility return negative value; hence, $U_n^F(\gamma_n, \gamma_{-n}, \tau_n)$ and $U_n^F(\gamma'_n, \gamma_{-n}, \tau_n)$ are increasing function of γ_n and γ'_n, respectively. Since $\gamma_n \geq \gamma'_n$ for all $n \in \mathcal{N}$, $U_n^F(\gamma_n, \gamma_{-n}, \tau_n) - U_n^F(\gamma'_n, \gamma_{-n}, \tau_n)$ returns always non-negative value and hence positivity is proved. Defining the incremental utility of player n when changing its action from γ_n to γ'_n as $\Delta U_n^F(\gamma_n, \gamma'_n, \gamma_{-n}, \tau_n) = U_n^F(\gamma_n, \gamma_{-n}, \tau_n) - U_n^F(\gamma'_n, \gamma_{-n}, \tau_n)$. With the proof of positivity and $\gamma_n \geq \gamma'_n$, $\Delta U_n^F(\gamma_n, \gamma'_n, \gamma_{-n}, \tau_n)$ returns always non-negative value and decreases with the increase in γ_{-n}. Therefore for all $\gamma_{-n} \geq \gamma'_{-n}$, $|\Delta U_n^F(\gamma_n, \gamma'_n, \gamma_{-n}, \tau_n) - \Delta U_n^F(\gamma_n, \gamma'_n, \gamma'_{-n}, \tau_n)| \leq 0$. Hence monotonicity is proved. To prove Scalability, we have $U_n^F(\kappa(\gamma_n, \gamma_{-n}, \tau_n)) - \kappa(U_n^F(\gamma_n, \gamma_{-n}, \tau_n)) = (\kappa - 1)(C_{n,s} + C_{n,a})$. Since $\kappa > 1$, Condition 3 of Lemma 2 is proved. □

3.4.2 Estimation of Fraction of Task Assigned to Each WM

$$U^L(\tau_n, \boldsymbol{\gamma}_n) = \sum_{n=1}^{N} \sum_{z=1}^{Z_n} \mathcal{P}_n(\boldsymbol{\gamma}) \zeta_{n,z} (1 - \alpha_n) Q_{n,z}. \tag{3.10}$$

$$\textbf{Problem 2:} \quad \min_{\tau_n} \quad U^L(\tau_n, \boldsymbol{\gamma}_n) \tag{3.11a}$$

$$\text{s.t.} \quad \sum_{n=1}^{N} \sum_{z=1}^{Z_n} \pi_{n,z} \tau_n \mathcal{D} = 1, \tag{3.11b}$$

$$\max_{n \in \mathcal{N}} \mathbb{T}_n < \beta, \tag{3.11c}$$

$$0 \leq \tau_n \leq 1. \tag{3.11d}$$

where $U^L(\cdot)$ be the utility function of MM, constraints indicate the total summation of all fractions of task equal to unity, and every WM completes its task within the given deadline.

Theorem 3.2 *The optimal task assigned to the WM by MM for executing the task is:*

$$\tau_n^{\star} = \frac{\lambda_{n,2}}{\hat{a}_n} \times \frac{\hat{b}_n(1-\gamma_{-n})Q_{n,z} + \mathcal{P}(\boldsymbol{\gamma}_n)\hat{Q}_{n,z}}{\Delta_{n,1} - \mathcal{P}(\boldsymbol{\gamma}_n)\hat{b}_n Q_{n,z}},$$

$$\textit{where } \Delta_{n,1} = \frac{\pi_{n,z}\lambda_{n,2}(\hat{b}_n(1-\gamma_{-n})Q_{n,z} + \mathcal{P}(\boldsymbol{\gamma}_n)\hat{Q}_{n,z})}{\hat{a}_n(1 - \sum_{i \neq n}\sum_{z=1}^{Z_i} \beta_{i,z}\tau_i)} + \mathcal{P}(\boldsymbol{\gamma}_n)\hat{b}_n Q_{n,z},$$

$$\hat{a}_n = a(1-\alpha_n)(2-\delta-\delta\gamma_{-n}), \hat{b}_n = \pi_{n,z} D(1-\alpha_n).$$

Proof: Eq. 3.11 gives the unique and near-optimal best response strategies of the WMs. Using backward induction method, Eq. 3.11 can be rewritten as:

$$\max_{\tau_n} \quad -\sum_{n=1}^{N}\sum_{z=1}^{Z_n} \mathcal{P}_n^{\star}(\boldsymbol{\gamma})\zeta_{n,z}(1-\alpha_n)Q_{n,z}^{\star}$$
$$s.t. \ \sum_{n=1}^{N}\sum_{z=1}^{Z_n} \tau_n = 1, \max_{n\in\mathcal{N}} \mathbb{T}_n < \beta, \ \textbf{and} \ 0 \leq \tau_n \leq 1.$$

To prove the uniqueness of the best response strategy, we use the Hessian matrix and second-order partial derivative of $U^L(\tau_n, \boldsymbol{\gamma}_n)$ *w.r.t.* τ_n and $\tau_{\bar{n}}$. The diagonal elements of the Hessian matrix are negative, and off-diagonal elements are zero. Therefore, the Hessian matrix of $U^L(\tau_n, \boldsymbol{\gamma}_n)$ is strictly negative definite, which implies that **Problem 2** is a standard convex maximization problem; hence, proved the uniqueness. The optimal solution of **Problem 2** can be obtained by KKT conditions, and the Lagrangian is given as:

$$\tau_n^{\star} = \frac{\lambda_{n,2}}{\hat{a}_n} \times \frac{\hat{b}_n(1-\gamma_{-n})Q_{n,z} + \mathcal{P}(\boldsymbol{\gamma}_n)\hat{Q}_{n,z}}{\Delta_{n,1} - \chi_{n,z}\Delta_{n,2} - \mathcal{P}(\boldsymbol{\gamma}_n)\hat{b}_n Q_{n,z}} \tag{3.12}$$

where, $\hat{a}_n = a(1-\alpha_n)(2-\delta-\delta\sum_{i\neq n}\sum_{z=1}^{Z_i}\gamma_{i,z})$, $\hat{b}_n = \pi_{n,z}D(1-\alpha_n)$, and $\chi_{n,z} = \frac{D^2\mu_{n,z}}{\mathcal{R}_n L_n}$. By putting value of τ_n into the constraint of **Problem 2**, we get:

$$\Delta_{n,1} = \frac{\pi_{n,z}\lambda_{n,2}(\hat{b}_n(1-\gamma_{-n})Q_{n,z} + \mathcal{P}(\boldsymbol{\gamma}_n)\hat{Q}_{n,z})}{\hat{a}_n(1-\sum_{i\neq n}\sum_{z=1}^{Z_i}\beta_{i,z}\tau_i)} + \mathcal{P}(\boldsymbol{\gamma}_n)\hat{b}_n Q_{n,z}.$$

Substituting the value of $\Delta_{n,1}$ from Eq. 3.13 and $\Delta_{n,2} = 0$ into Eq. 3.12 and hence proved. □

3.4.3 Implementation of the Solution

This section proposes a semi-distributed algorithm to implement the solution of (α, β)-**TC** problem. MM allocates the fractions of a task to the WMs on the round basis *i.e.,* computation on the WM is locked for one round. Later, the fractions of the task

(allocated to WMs) are computed again upon the change in resources of WMs. Let t denote the time duration for one round. In the first round, γ_n, and τ_n are computed by using Theorem 3.1 and Theorem 3.2, respectively. $\forall n \in \mathcal{N}$ with the security constraint α, WMs execute the task based on the computed γ_n and τ_n for the current round. In the next round, WMs check their α_n and $\boldsymbol{\mu}_n$, and send the information to the MM. If there is any change in new α_n or $\boldsymbol{\mu}_n$, MM again computes τ_n using Theorem 3.2 and α_n and $\boldsymbol{\mu}_n$.

Algorithm 3.1: Container-based task offloading system.

Input: $\mathcal{D}, \gamma_{max}, \alpha, \beta, \boldsymbol{\mu}_n$;
Initialization: Rounds, $\alpha_n^0 = \alpha_n^1 = \boldsymbol{\mu}_n^0 = 0$, $\boldsymbol{\mu}_n^0 = \boldsymbol{\mu}_n$;
for $k \leftarrow 1$ *to Rounds* **do**
  for $n \leftarrow 1$ *to* N **do**
    if $(\alpha_n^k != \alpha_n^{k-1} \;||\; \boldsymbol{\mu}_n^k != \boldsymbol{\mu}_n^{k-1})$ **then**
      Check α_n^k ;
      Calculate $\boldsymbol{\gamma}_n^k$ using Theorem 3.1;
      Calculate τ_n^k using Theorem 3.2;
    return $(\boldsymbol{\gamma}_n^k, \tau_n^k)$;
    $\alpha_n^{k+1} = \alpha_n$;
    $\boldsymbol{\mu}_n^{k+1} = \boldsymbol{\mu}_n$;
return $\boldsymbol{\gamma}_n, \boldsymbol{\tau}_n$;

3.5 Empirical Evaluation

This section empirically evaluates the performance of the proposed (α, β)**-TC** problem. The section covers the experimental setup, followed by experimental results.

3.5.1 Experimental Setup and Baseline Schemes

The experimental setup in this work is inspired by the work in [**?**]. We utilized the Docker swarm as the container manager engine and implemented the approach in Python 3.7.3. Each container is deployed on a machine allocated with limited CPU

resources. The simulation is performed on the HP Prodesk desktop with 8-core 3.2 GHz CPU, 24 GB RAM with the installed operating system Linux (Ubuntu 20.04). The simulation results depicted in the subsequent sections are averaged over 500 tests. To facilitate the ablation studies of the proposed approach, we have considered four schemes denoted as $\mathbf{S_1}$, $\mathbf{S_2}$, $\mathbf{S_3}$, and $\mathbf{S_4}$ (proposed), illustrated in Table 3.2.

Table 3.2: Illustration of baseline schemes for ablation studies, where N is the number of WMs.

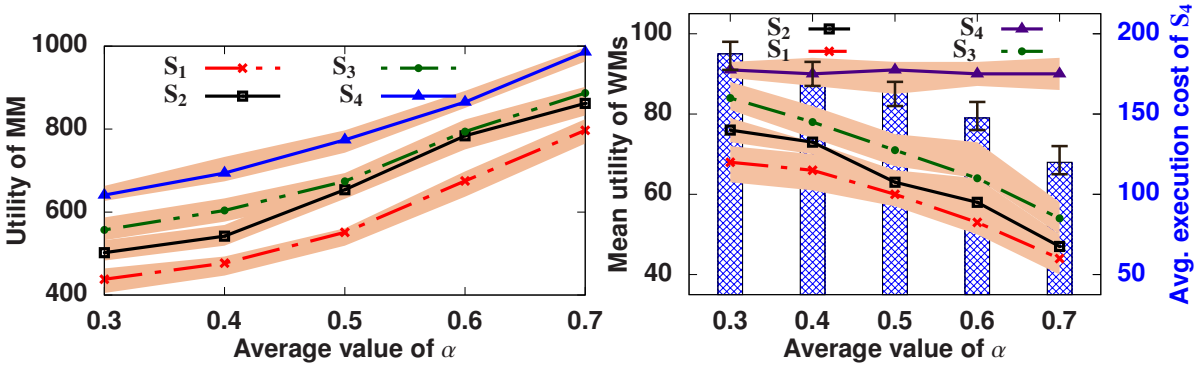
Schemes	**Properties**	**WM(s)**	**Task fraction (τ)**
$\mathbf{S_1}$	No task partition	1	$\tau = 1$
$\mathbf{S_2}$	Equal task partition	N	$\tau = \tau_1 + \tau_2 \cdots \tau_N$, where, $\tau_1 = \tau_2 = \cdots = \tau_N = 1/N$.
$\mathbf{S_3}$	Random task partition	N	$\tau = \tau_1 + \tau_2 \cdots \tau_N$, where, $\tau_1, \cdots, \tau_N$ are randomly chosen.
$\mathbf{S_4}$ **(proposed)**	Deterministic task partition	N	$\tau = \tau_1 + \tau_2 \cdots \tau_N$, where, $\tau_1, \cdots, \tau_N$ are deterministically assigned by optimizing cost and security within given deadline.

3.5.2 Experimental Evaluation

The utility metric of MM (U^L) is the price paid to the WMs for executing the task, the expression for estimating U^L is given in Eq. 3.10. The utility of each WM is the profit or loss achieved by the WM upon executing the task fraction received from MM. We can estimate the utility of i^{th} WM (U_i^F) using Eq. 3.5. Later, these performance metrics estimate the fairness of task fraction distribution over WMs from MM. A high fairness index implies high system efficiency. To determine the fairness index (F^I), we consider Jain's fairness index, defined as $F^I = \frac{\left(\sum_{i=1}^{N} t_i\right)^2}{N.\sum_{i=1}^{N} t_i^2}, \forall i \in N$, where, t_i is the execution time of task fraction τ_i on i^{th} WM. The required execution time utility metric of Scheme S_j (R_j^E) is given by $R_j^E = \frac{E(\mathbf{S_j}) - E(\mathbf{S_4})}{E(\mathbf{S_4})} \times 100$, where, $E(\cdot)$ is the function that determine execution time.

3.5.2.1 Impact of α

This work experimented to study the impact of α on the utilities of MM and WMs. We assumed the same deadline for all the schemes during the experiment and set the datasize to a fixed value of 1 GB. Fig. 3.2 (a) demonstrated the utility of MM (U^L) increases with the increase in α, given a fixed value of α on each WM. The result depicted, Scheme $\mathbf{S_4}$ outperforms others schemes ($\mathbf{S_1} - \mathbf{S_3}$). It is because $\mathbf{S}_4$ used the cost and security optimization mechanism, which improved utility. Moreover, the increase in α reduced the cost paid by MM to WMs, resulting in utility improvement of MM. $\mathbf{S_1}$ gain the least improvement in the utility of MM due to the absence of a mechanism for task distribution on multiple WMs. As MM provides a low price for a higher α; thus, the average utility of WMs (U^F) decreases with an increase in α, as shown in Fig. 3.2(b).



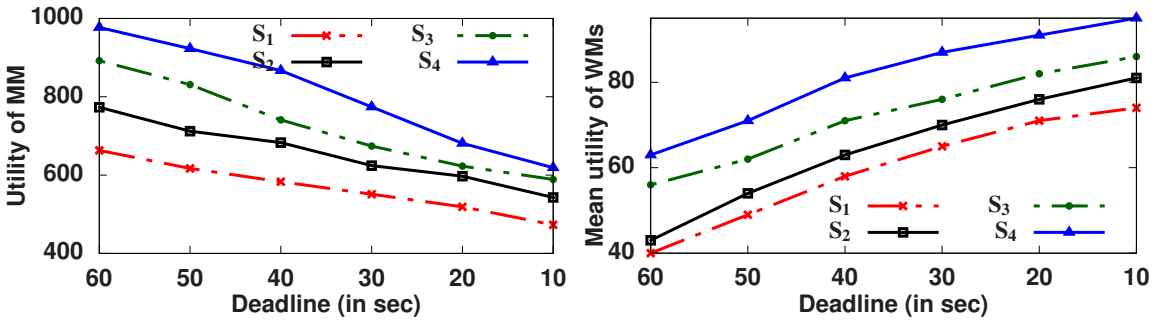
(a) U^L *vs.* average value of α. (b) U^F *vs.* average value of α.

Figure 3.2: Impact of security compromise (α) on utility of MM and average utility of WMs.

3.5.2.2 Impact of Deadlines

We next illustrated the impact of the task deadline on the utility of MM and WMs under different schemes ($\mathbf{S_1} - \mathbf{S_4}$). We considered the datasize of 1 GB and different task fractions on WMs. Fig. 3.3(a) shows the improvement in U^L with the increase

in deadline. As the deadline increased, MM assigned the task to the slower machines, demanding a low execution cost. Further, the execution of the task on a single machine in $\mathbf{S_1}$ did not provide much room for improving the average utility of WMs. Therefore, after some initial improvement in average utility in $\mathbf{S_1}$, the utility remains constant with an increase in the deadline, as illustrated in Fig. 3.3(b). Additionally, in $\mathbf{S_1}$, the profit of a single node is much higher than other schemes because of complete task execution. Moreover, utility of $\mathbf{S_4}$ outperforms $\mathbf{S_2}$ and $\mathbf{S_3}$ on given deadline, due to cost optimization. We can observe from this result that the increment in deadline increases system utility, *i.e.,* summation of utilities of MM and WMs.



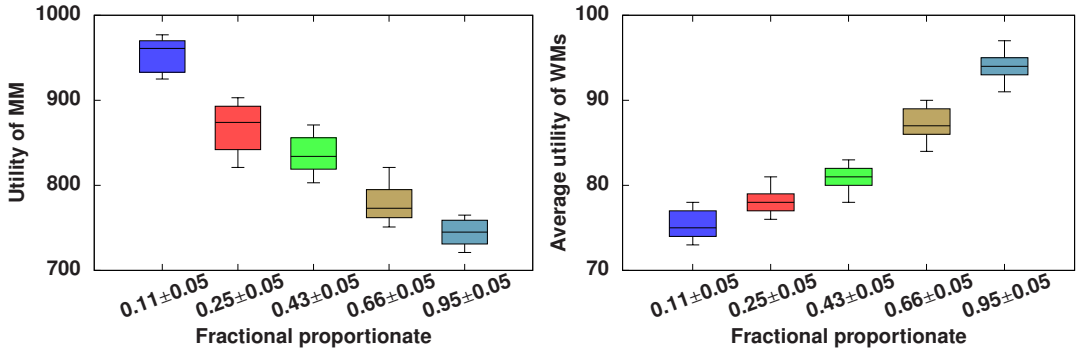
(a) U^L *vs.* deadline. (b) U^F *vs.* deadline.

Figure 3.3: Impact of the deadline on the utility of MM and utility of WMs.

3.5.2.3 Impact of Fractional Proportionate

This experiment studied the impact of the fractional proportionate matrix on the utility of MM and the average utility of WMs in $\mathbf{S_4}$. Fractional Proportionate (FP) is defined as: $FP = \frac{\min\{\tau_1,\tau_2,\cdots,\tau_N\}}{\max\{\tau_1,\tau_2,\cdots,\tau_N\}}$. Fig. 3.4(a) illustrates the decrement in the utility of MM with the increase in the fractional proportionate. This is because the proposed approach achieved an optimal condition when the task distribution achieved higher utility within the given deadline. In other words, the fraction of tasks assigned to high-price WMs is always less than the task assigned to low-price WMs. Thus, if fractional proportionate ≈ 0.1, indicating the task assigned to low-cost WMs is higher than high-cost WM,

it results in the high utility of MM. Similarly, at lower fractional proportionate, the average utility of WMs is low, in Fig. 3.4(b).



(a) U^L *vs.* FP. (b) U^F *vs.* FP.

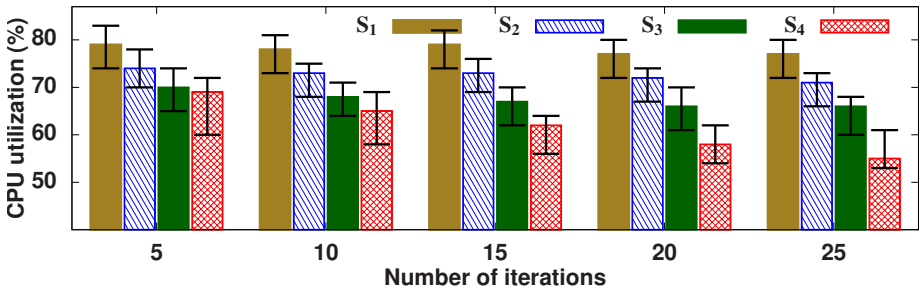
Figure 3.4: Impact of fractional proportionate on utilities.

3.5.2.4 Impact of Different Schemes on CPU Utilization and Memory Consumption

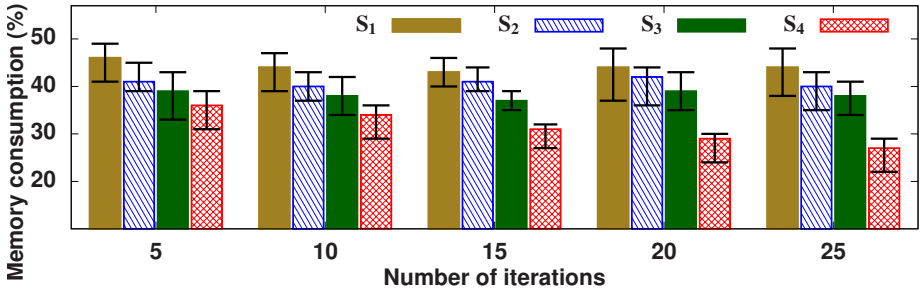
In this experiment, we studied the impact of different schemes ($\mathbf{S_1} - \mathbf{S_4}$) on CPU utilization and memory consumption. As one core of CPU with a clock speed of 1 GHz is assigned to each container on WMs. Therefore, maximum CPU utilization reached up to 80% for Scheme $\mathbf{S_1}$. The task's deadline is sufficient to accommodate 25 iterations of the proposed algorithm. Fig. 3.5 illustrates the impact of choosing different schemes on CPU utilization and memory consumption with the increase in the number of iterations. The result depicted that Schemes $\mathbf{S_1}$ and $\mathbf{S_2}$ do not change their CPU utilization because the task or its fraction assigned to WMs are fixed a prior. $\mathbf{S_4}$ estimated the possible optimal condition for minimizing CPU and memory consumption.

3.5.2.5 Impact of different schemes on task accomplishment

Finally, we studied the impact of different schemes ($\mathbf{S_1} - \mathbf{S_4}$) on the task accomplishment metric. Task accomplishment metric is the maximum task that can be allocated, as



(a) CPU utilization.



(b) Memory consumption.

Figure 3.5: Impact of different schemes ($\mathbf{S_1} - \mathbf{S_4}$) on CPU utilization and memory consumption.

per the given storage on the device. To perform this experiment, we considered seven different tasks $T_1 - T_7$, as shown in Table 3.3. The tasks include Locomotion Mode Recognition (LMR) [**?**], River Water Monitoring (RWM) [**?**], Vehicle Driving Behavior (VDB) [**?**], and Driver Behavior Detection (DBD) [**?**]. Versions I and II specified the different datasizes. LMR-U stands for LMR with unseen classes. We varied the CPU and memory availability at the host from 40% to 90%.

We have chosen the best possible combination of tasks ($T_1 - T_7$) to be executed on WMs on given resources and deadlines during the experiment. This work also sets the priority of tasks in chronological order, *i.e.,* T_1 is the highest priority task and T_7 is the lowest priority task. Next, we considered four machines (one MM and three WMs), each having allocated memory of 2 GB and one core of processor with a clock speed of 1 GHz. We studied the impact of task accomplishment using the following two cases:

1. **Case 1:** *Available CPU of host machine fixed to* 90%*, memory varies from* 40%-90% *:* Table 3.4 illustrated Scheme $\mathbf{S_1}$ is incapable of accommodating multiple

Table 3.3: Task specifications to analyze accomplishment metric.

Source	Year	Task	Task type	Datasize	Sensors	Classes count
Mishra *et al.* [?]	2020	T_1 T_2 T_3	LMR-I LMR-II LMR-U	1.5 GB 2.5 GB 1.5 GB	MPU-9250	6
Water-to-Cloud [?]	2021	T_4 T_5	RWM-I RWM-II	500 MB 1.0 GB	HI-9829	6
Zhang *et al.* [?]	2021	T_6	VBD	100 MB	MPU-6050	5
Yuksel *et al.* [?]	2021	T_7	DBD	250 MB	MPU-6050	4

tasks due to a single WM. The proposed Scheme $\mathbf{S_4}$ outperforms all the considered schemes at a varying percentage of available memory. It indicates that $\mathbf{S_4}$ can accommodate multiple tasks simultaneously. Though $\mathbf{S_2}$ and $\mathbf{S_3}$ also involve similar WMs but $\mathbf{S_4}$ facilitated optimal task allocation. The allocation provided an opportunity to fetch host machine resources.

Table 3.4: Illustration of impact of different schemes on task accomplishment with fixed CPU on all the worker machines.

		Available memory (in %)					
		40	**50**	**60**	**70**	**80**	**90**
Schemes	$\mathbf{S_1}$	0.50	0.0	0.50	0.25	0.34	0.43
	$\mathbf{S_2}$	0.67	0.5	0.75	0.75	0.67	0.72
	$\mathbf{S_3}$	0.67	0.5	0.75	0.75	0.67	0.72
	$\mathbf{S_4}$	1.0	0.5	0.75	0.75	0.84	0.85

2. **Case 2:** *Available memory of host machine fixed to* 90%*, CPU varies from* 40%-90% *:* Apart from Case 1, this case studied the impact of varying CPU availability on task accommodation with fixed memory (90%). Table 3.5 illustrated the impact of different schemes on task accomplishment when available memory is fixed, and CPU resource is increasing. Similar to Case 1, here also Scheme $\mathbf{S_4}$ outperformed other schemes due to optimized task partitioning.

Table 3.5: Illustration of impact of different schemes on task accomplishment with fixed memory on all the worker machines.

		Available CPU (in %)					
		40	**50**	**60**	**70**	**80**	**90**
Schemes	$\mathbf{S_1}$	0.33	0.0	0.25	0.25	0.43	0.28
	$\mathbf{S_2}$	0.67	0.5	0.75	0.75	0.67	0.72
	$\mathbf{S_3}$	0.67	0.5	0.75	0.75	0.67	0.72
	$\mathbf{S_4}$	1.0	0.5	1.0	0.75	1.0	1.0

3.6 Conclusion and Discussion

This chapter proposed an IIoT task containerization approach for securely executing the task within the maximum allowable response time. Unlike earlier work, the proposed system is suitable for industrial applications, where both security and deadline constraints persist side-by-side. The system has used the cost paid by each machine to execute and secure the task. We next built a Stackelberg game, where workers and master machines are the followers and leaders, respectively.

We validated the proposed approach through mathematical analysis and experimental evaluations. We also verified our approach against existing approaches in similar domains. Furthermore, the proposed approach assumed the ideal case where the WMs refused to execute the assigned fraction of the task. It can be easily extended for the scenario where WMs can refuse to perform the allocated fractions. Additionally, the approach partitioned the task, assuming that each partition is independent and executed disjointly. The approach can be easily extended when the fractions possess inter-dependencies.