International Journal of Recent Technology and Applied Science
p-ISSN: 2721-2017, e-ISSN: 2721-7280

Original Research Paper

Automatic Task Provisioning and Routing Framework for Carrier Robots in Smart Factory with Edge Computing

Philip Tobianto Daely[1*], Oktavia Ayu Permata[1], Bernadus Anggo Seno Aji[1]

[1] *Department of Information Technology, Telkom University, Surabaya, Indonesia.*

Article History
Received:
29.10.2024

Revised:
14.11.2024

Accepted:
27.11.2024

***Corresponding Author:**
Philip Tobianto Daely
Email:
ptdaely@telkomuniversity.ac.id

Abstract: This paper proposed a pickup and delivery (P&D) task provisioning and vehicle routing framework for carrier robots in a factory with edge servers assisting as the intermediary between robots and working stations, where the task requests come. The carrier robots must pick up and deliver each assigned load from and to designated locations with minimal traveled distance and without violating designated constraints. Edge servers are utilized to facilitate communication between the main server and delivery robots and assist the main server in deciding the best robot for each incoming task request and the updated route to facilitate execution of tasks by each carrier robot. The problem of pickup and delivery for each robot is modeled based on Dial-a-Ride Problem, and a discrete bio-inspired algorithm is proposed to solve this problem. The tasks are distributed to edge servers and carrier robots by taking their service loads into account. The simulation results show that the proposed framework can provide an effective solution towards optimizing the pickup and delivery process in a smart factory.

Keywords: Dial-a-Ride-Problem, Edge Computing, Particle Swarm Optimization, Pickup and Delivery, Smart Factory.

1. Introduction

The fourth industrial revolution, also known as Industry 4.0, has brought the focus of experts into making manufacturing industries smart and connected to the internet. These interests appear because of demands for flexibility and ease to monitor production processes and management in factories. The stakeholders believe that having extensive connectivity and intelligence, the effectiveness and efficiency of factories may produce a higher quality of output and outcomes. Factory logistics, especially the item distribution between working stations, is one process that can be optimized to achieve the concept of a smart factory. The automatic distribution process could boost the performance of the factory.

One of the essential processes in a factory is logistics, especially the distribution of items (such as materials and products) between locations or stations inside the factory area. There are many ways of distribution method used in factories, starting from using human workers, human-operated carriers (such as trolleys, mov- able racks, or forklifts) to autonomous carriers or robots [1]. This evolution from manual to autonomous approach is deemed necessary for increasing the productivity of factories because autonomous carriers can carry heavy loads, do not require a long time to rest, capable of maintaining performance consistency, and insusceptible from fatigue, illness, or death, hence low operating cost [2]. The deployment of autonomous carriers also helps managements to focus their human resources on tasks that require higher intelligence, more delicate execution, and broader room for improvisation, if required.

The problem of robots performing pickup and delivery can be modeled as a Dial-a-Ride Problem (DARP). DARP has mostly been used to model demand- responsive transport systems, where such a system usually provides an unconventional means of public transportation [3], [4]. DARP is considered an NP-hard problem, which makes it very difficult to solve using exact algorithms. Recently, the option to use metaheuristic algorithms has gained interest for their relative ease of design and efficiency in getting an immediate solution. The work in [5] proposes a Regional Multi-Objective Tabu Search algorithm for a green heterogeneous Dial- a-Ride problem. The proposed algorithm utilizes Tabu search and multi-objective optimization techniques to find solutions that minimize the total distance traveled, waiting times, and environmental impact. Another work in [6] used Large Neighborhood Search (LNS) algorithm to solve DARP in ride-hailing services. A neural network model is introduced and trained to generate the best values for parameters of LNS. The work in [7] proposes an Ant Colony Optimization (ACO) algorithm for solving the Dial-a-Ride Problem (DARP) with time windows and capacity restrictions. The algorithm includes a penalty function that discourages violations of the capacity and time window constraints.

Besides carrier robot fleet management and task pro- visioning, there are many more processes occurring and being executed in smart factory, which may lead to concentrated computational burden at the main server. For carrier robots to navigate through the work area and avoid obstacles, choices must be made fast and reliable. Edge computing is a concept of distributed computing that is generally utilized with the objective of reducing latency by processing data near the source [8]. The cloud-edge joint scheme allows the robots to use the edge's low-latency processing capability to make quick decisions while also utilizing the cloud's computing resources for more difficult tasks [9]. By utilizing edge computing, the time required to move items can be reduced and routes can be optimized for carrier robots. Edge computing also increases reliability, by having redundancy in case one of the edge servers is down or having connectivity problem [10]. As they move across the factory floor, carrier robots generate a large amount of data. This data can be collected and processed in real-time using the cloud-edge joint scheme, allowing factory to make better production and logistics decisions [11].

In this paper, a P&D task assignment framework for industrial factories is proposed. The framework comprises of three main components: carrier robots, edge servers, and main server. The carrier robots are the ones carrying out the physical actions of pickup and delivery. They are also equipped with sensors for positioning and measurements. The data from sensors are transmitted to edge servers to determine the optimum route to take for each robot. The tasks are originated by stations in the factory and gathered at the main server (cloud). The main server then will issue orders to edge servers to assign each task to the appropriate robot. The main server also acts as central data storage, storing all data from edge servers, which can later be used for maintenance.

The contributions of this paper are listed as follows:

1) A framework for a smart factory is proposed to facilitate automatic provisioning of P&D tasks to multiple carrier robots and data gathering at the main server.

2) Utilization of edge servers as intermediaries between the main server and carrier robots is considered to reduce the number of messages coming in and out of the main server.
3) A task insertion and routing algorithm based on Discrete Particle Swarm Optimization (DPSO) is developed and utilized by edge servers to determine the most suitable carrier robots for a P&D task.

The rest of this paper is organized as follows. Section 2 provides a review of previous works related to this paper. Section 3 gives the problem formulation and presents the proposed framework to tackle the problem. Section 4 discusses the performance of the proposed framework in simulations. Finally, Section 5 concludes the paper.

2. Literature Review

The frameworks for utilizing distributed computing (e.g., edge, fog, and cloud computing) for industrial robotic applications have been studied during recent years. In this section, we elaborate in detail the existing frameworks related to the distributed computing for industrial robot applications. First, we distinguish the advantages of distributed computing concepts for industrial robot applications. Then, we disseminate the use of distributed computing paradigms from the traditional approach (i.e., cloud computing) to the latest trends (i.e., edge and fog computing) for the industrial robot applications. Furthermore, we compare the existing frameworks with our proposed mechanism.

The recent work in [12] proposed a task offloading mechanism that can achieve an optimal energy-efficient for cloud-based multi-robot systems. The proposed mechanism did not only consider the load balancing between robots but also between robot and cloud, as for utilizing the resources. The reason was that the task offloading scheme for the robot was more complicated than the other applications due to the mobility of a robot. Also, they presented an integrated framework to enable their proposed task offloading mechanism.

Different from the approach mentioned above, which utilizes cloud computing, some have conducted research to adopt edge computing for industrial robot applications. The study in [13] proposed a framework named iRobot-factory to enable the intelligent robot factory. The proposed framework comprised of two aspects, which are based on cognitive manufacturing and edge computing. They further examined the performance of the proposed framework through experimental study and compared it with the traditional factory scheme.

The authors in [14] investigated the optimal resource allocation for edge and cloud-based robot in the smart factory. They addressed the diverse energy consumption and cost of executing the task for a robot in a distributed manner as for the problems. Thus, the proposed resource allocation mechanism was modeled as a constrained multi-objective optimization problem. Furthermore, they adopted a multi-objective evolutionary method named Non-Dominated Sorting Genetic Algorithm-II (NSGA-II) algorithm to solve it. This work focuses more on resource allocation and energy saving as objective, whereas our proposed framework strives to effectively carry out pickup and delivery in smart factory by minimizing travel distance of carrier robots.

The work in [15] proposes a heuristic planning and scheduling method for multiple automatic guided vehicles (AGVs) in logistics systems based on a cyber-physical system. Task planning, vehicle scheduling, and path planning are the three stages of the proposed approach. Tasks are assigned to AGVs during the task planning stage based on their availability and capacity. Based on current traffic conditions and battery life, the vehicle scheduling stage determines the sequence of tasks to be performed by each AGV. The path planning stage determines the best path for each AGV to take to complete their assigned tasks. The proposed method is tested on a real-world logistics system, and the results show that it outperforms traditional heuristic-based methods in terms of reducing travel time and increasing logistics system efficiency. Our proposed framework differs from this work by focusing more on the determining routes of carrier robots to pick and deliver items from and to multiple target stations. The target stations to be visited are determined from Task Insertion Algorithm with the objective of minimizing travel distance while considering workload of carrier robots and edge servers.

The work in [16] proposes a network intelligence- based approach for controlling industrial robots in an F- RAN environment. To optimize the control of industrial robots, the approach employs artificial intelligence techniques such as machine learning and deep reinforcement learning. The proposed system is divided into three layers: perception, decision-making, and action. The perception layer collects data from sensors and cameras, which the decision-making layer processes in real time based on task

requirements and the environment. The decision is then carried out by the action layer, which controls the robot's movements. The proposed method is tested on a real-world dataset, and the results show that it outperforms traditional control methods in terms of scheduling efficiency and mitigate backhaul traffic problems. Similar with this work, our proposed framework also implements task provisioning for carrier robots, but instead of using machine-learning based method, we opted to develop a DPSO based Task Insertion Algorithm, which does not require gathering dataset for training, and can provide immediate results during operation.

The work in [17] proposes a two-tier model predictive control (MPC) architecture that is aided by edge computing for the navigation of automatic guided vehicles (AGVs) in an industrial setting. The proposed approach is divided into two parts: the local tier and the global tier. The local tier oversees controlling the AGVs' motion and monitoring the local environment, whereas the global tier oversees optimizing the AGVs' trajectories and avoiding AGV conflicts. Edge computing is used by the global tier to improve MPC performance by reducing the computational load on the AGVs. The results show that the proposed approach outperforms traditional MPC methods in terms of reducing AGV travel time and improving navigation system efficiency. Although the proposed architecture is intended for implementation in an industrial setting, there is not much consideration of industrial scenario or parameters involved in the presented simulation. Our proposed framework instead shows the scenario of carrier robots performing pickup and delivery between workstations in a smart factory.

A distributed computing architecture for logistic job allocation in a smart factory is proposed in work [18]. The proposed architecture is made up of multiple agents that communicate with one another to assign logistic jobs to various machines and resources in the factory. A multi-criteria decision-making algorithm is used by the agents to optimize job allocation based on factors such as machine availability, job priority, and resource capacity. The proposed approach is tested using a simulation model, and the results show that it outperforms traditional centralized job allocation methods in terms of reducing overall job completion time and improving the logistic job allocation system's efficiency. This work uses Binary Particle Swarm Optimization as the base of AGV routing algorithm, where the search agents iteratively regenerate particles (solutions), each containing assignments of tasks to AGV in form of binary value in particle's elements and will determine the best solution which has the shortest travel distance. This method requires additional steps to solve subproblem of routing each AGV to every stations required to visit. Whereas the DPSO based Task Insertion Algorithm in the proposed framework uses different mechanisms, in which the particle's elements contains a discrete positive integer value reflecting sequence of visit to workstations, thus does not require additional algorithm for routing.

3. Methodology

3.1. Problem Formulation of DARP for Carrier Robots in Smart Factory

The problem of carrier robot executing P&D tasks is modeled as DARP. In an industrial factory, there is a set of workstations $\boldsymbol{s} = s_1, s_2, \dots, s_S$ with $S = |\boldsymbol{s}|$ and station s_1 as the robot pool, where all robots stay when no task given to them. The distance between two stations is depicted as $d_{i,j}$, representing the traveling distance between station s_i to station s_j. Let a robot with a set $\boldsymbol{r}$ of $R = |\boldsymbol{r}|$ assigned tasks. Each task is executed by picking up the load from an origin station and delivering it to a destination station. For the robot to get the best travel route, an objective function used to solve the problem is formulated as follows:

$$\min \sum_{i=1}^{S}\sum_{j=1}^{S} d_{s_i,s_j}\, x_{s_i,s_j}, \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{S} x_{s_i,s_j} \le 1, \quad \forall i \in 1,2,\dots,S,$$

$$\sum_{i=1}^{S} x_{s_i,s_j} \le 1, \quad \forall j \in 1,2,\dots,S,$$

$$t_A < t_o \leq t_B, \quad \forall r \in r,$$
$$t_{des} > t_{ori}, \quad \forall r \in \boldsymbol{r},$$
$$L \leq L_\theta, \quad \forall s \in \boldsymbol{s},$$
$$\Upsilon \leq \Upsilon_\theta, \quad \forall r \in \boldsymbol{r},$$

with x_{s_i,s_j} equals to zero if there is no direct link from station s_i to station s_j or one if there is a direct link in the full route. The first constraint and the second constraint ensure that the robot departs from and goes to each station only once. The third constraint implies for each task r, the pick up time at its origin station t_o must be after the time of task issuance t_A and before or equal to task pick up deadline $t_B = t_A + t_{wait}$, with t_{wait} as the maximum waiting duration for task to be picked up. The fourth constraint denotes for each task r, the destination station visiting time t_{des} must be after the pickup time at the origin station t_{ori}. The fifth constraint imposes that at every station s, the robot's load L must not exceed the set maximum capacity L_θ. Lastly, the sixth constraint implies that the detour ratio Υ for each task must be under the limit Υ_θ (see details in (5)). Equation (1) is used when the assigned tasks for each robot for one round of travel (depart from and go back to robot pool) are issued before departing from the robot pool only. This equation is not appropriate with the scenario in this paper, where each robot may also be assigned a new task while still at the robot pool and during travel, thus adjustments are needed.

At each time-step, each robot may receive one task and will be in one of three states:

- staying at the robot pool ($z = 1$)
- staying at another station ($z = 2$)
- moving to a station ($z = 3$)

with $z \in \{1, 2, 3\}$ denotes the robot's state. When $z = 1$, the robot currently does not have any assigned task, which makes the incoming new task directly assigned, and the travel route determination will be simple, from the robot pool to the task's origin station, then to the task's destination station, and go back to robot pool. When receiving a new task while $z = 2$ or $z = 3$, the robot will try to insert the new origin station and new destination station into the current route to travel. The Task Insertion Algorithm will try to determine the best new route so it will have the optimum travel distance, or in other words, the route with the least travel distance possible while still not breaking the constraints.

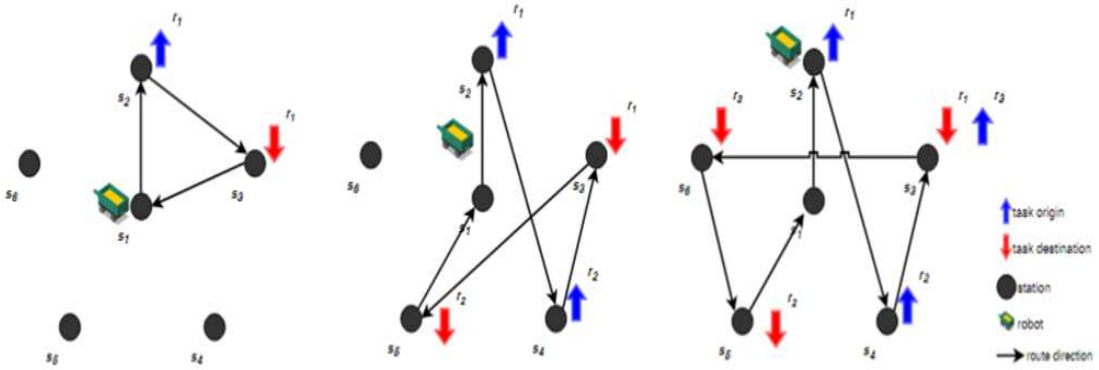

Figure 1. Illustrations of a Robot's Travel Route Update in Different States
(a) The Travel Route When Receiving a New Task r_1 while Staying at The Robot Pool s_1, with the New Task's Origin Station at s_2 and Destination Station at s_3
(b) The Travel Route When Receiving a New Task r_2 while Traveling from s_1 to s_2, with the New Task's Origin Station at s_4 and Destination Station at s_5
(c) The Travel Route When Receiving a New Task r_3 while Staying at s_2, with the New Task's Origin Station at s_3 and Destination Station at s_6

For example, in a time step, a new task is assigned to a robot. The new task has an origin station s_2 and destination station s_3. When the robot is in the first state, the new full route is $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_1$, as shown in Figure 1a. This full route is further divided into two lists: previous-stations list and next-stations list. The previous-stations list contains an ordered list of stations from the full route that are previously and currently being visited. The next-stations list contains an ordered list of stations from the full route that will be visited next. Given the new full route and the current position of robot at station s_1, the previous-stations list contains s_1 only and the next-stations list contains s_2, s_3, and s_1 in this order.

In the following time-step, the robot is in the second state, traveling from station s_1 to station s_2. In this time-step, a second new task is assigned with origin station s_4 and destination station s_5. The new full route needs to be determined to accommodate the second new task. The Task Insertion Algorithm will insert the new origin-destination station pair in the full route, and only the order of stations in next-stations list can be modified because the stations in previous-stations list have been visited. In next-stations list, the order of station s_2 and s_1 are also not modified because the robot is currently en route to station s_2 and final station is always station s_1. Assume that Task Insertion Algorithm outputs the new order of next-stations list: s_2, s_4, s_3, s_5, and s_1, which makes the new full route becomes $s_1 \rightarrow s_4 \rightarrow s_2 \rightarrow s_3 \rightarrow s_5 \rightarrow s_1$, as shown in Figure 1b.

After several time-steps, the robot finally reaches station s_2 and stays there for some time steps, which makes previous-stations list is updated to contain s_1 and s_2, also next-stations list is updated to contain s_4, s_3, s_5, and s_1. In this time step, the robot is in the third state. When the robot receives a third new task, similar process happens as in the second state, except the first station in next-stations list (station s_4) is also included in order modification. The third new task is assumed to have an origin station s_3 and destination station s_6. The Task Insertion Algorithm then outputs a new order for next-station list: s_4, s_3, s_6, s_5, and s_1. The new full route then becomes $s_1 \rightarrow s_2 \rightarrow s_4 \rightarrow s_3 \rightarrow s_6 \rightarrow s_5 \rightarrow s_1$, as shown in Figure 1c, with current robot position at station s_2.

Let $\mathbf{q_1} = \{q_{1,1}, q_{1,2}, \dots, q_{1,Q_1}\}$ be a set of stations in the previous-stations list and $\mathbf{q_2} = \{q_{2,1}, q_{2,2}, \dots, q_{2,Q_1}\}$ be a set of stations in the next-stations list of a robot, with $Q_1 = |\mathbf{q_1}|$ and $Q_2 = |\mathbf{q_2}|$. Based on the previous example, when the robot is travelling and receives a new task with its origin and destination station, the Task Insertion Algorithm will find the best order of modified next-stations list in the form as $q_2 = \{q_{2,1}, \dots, q_{2,x_1}, \dots, q_{2,x_2}, \dots, q_{2,Q_2}\}$, with x and y as the index of the new task's origin and destination station respectively, using objective function as follows:

$$\min \quad D(q_{1,Q_1}, q_{2,1}) + \sum_{k=1}^{Q_2-1} D(q_{2,k}, q_{2,k+1}), \tag{2}$$

$$\begin{aligned} \text{s.t.} \quad & z - 1 \le x < Q_2 - 1, \\ & \alpha < y \le Q_2 - 1, \\ & t_A < t_o \le t_B, \quad \forall r \in \boldsymbol{r}, \\ & t_{des} > t_{ori}, \quad \forall r \in \boldsymbol{r}, \\ & L_q \le L_\theta, \quad \forall q \in \boldsymbol{q_2}, \\ & \Upsilon \le \Upsilon_\theta, \quad \forall r \in \boldsymbol{r}, \end{aligned}$$

where the objective function basically calculates the total distance from the last visited station q_{1,Q_1} to the last station in the travel route $q_{2,Q_2} = s_1$, with $D(s_i, s_j) = d_{s_i,s_j}$. The first and second constraint allows x and y to be from $z - 1$ to $Q_2 - 1$ and the new task's origin must always precede the destination. The four remaining constraints are basically the same with the last four constraints of the objective function in (1). If the new task violates any of the constraints, the new task will be immediately rejected.

3.2. Proposed Automatic Task Provisioning and Routing Framework for Carrier Robots

The overall architecture of the proposed framework is shown in Figure 2. It consists of three layers: operation layer, edge layer, management layer. Data exchange can be performed intralayer. Interlayer data exchange can also be performed by adjacent layers. The Operation Layer consists of the carrier robots that execute the tasks given to them and the stations that request the task to the main server (cloud). The Edge Layer consists of edge servers, which are infrastructures that provide a connection between Management Layer and Operation Layer and perform task allocation and scheduling for all

carrier robots. This layer helps the main server in saving computation resources and costs. The Management Layer consists of the main server, management interface, and management team. The main server stores all data from edges servers and provides information and control of the framework to the management team via a management interface.

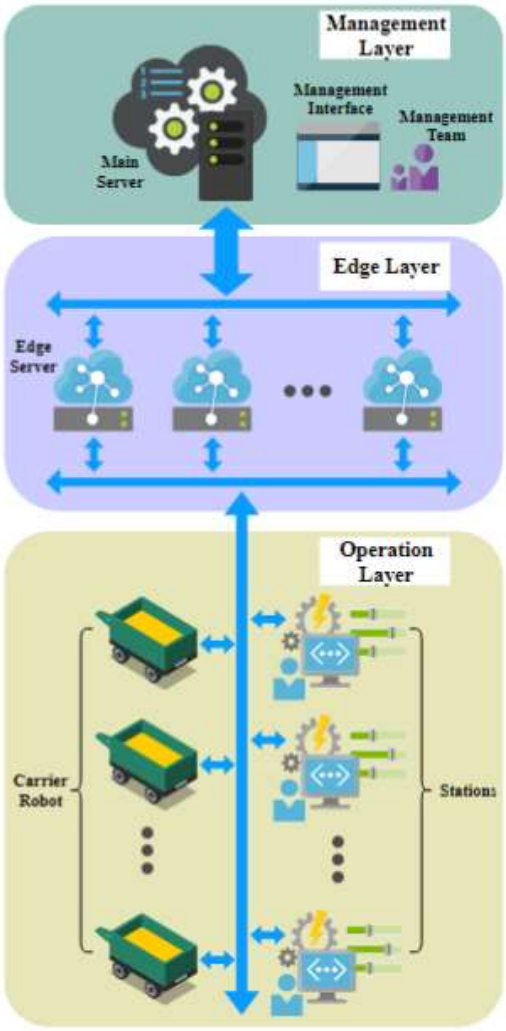

Figure 2. Architecture of Proposed Framework for Industrial Factory

The message flow is illustrated in Figure 3. The work begins with the Operation Layer. The stations will request P&D tasks for items with a certain amount and a delivery deadline. The task requests are sent to the main server via edge servers. The main server then puts the tasks in a queue on a First In, First Out (FIFO) basis. The main server will assign each task to be processed further by an edge server with the least service load. Each edge server will select an appropriate carrier robot with an appropriate service load. Finally, the carrier robots will execute the P&D tasks assigned to them and navigate through the area to reach every destination.

While navigating over the area, carrier robots also check for the traveling cost between locations. All carrier robots are equipped with a positioning device and sensors to measure their speed and orientation. These data are used to calculate the travel cost. The measured cost is used to update the edge servers about the changes in the area. If for some reason, an obstacle appears on the path between two locations and resulting in higher travel cost for the path, the edge servers will be able to adjust to this change and provide a route that is best for this situation.

The carrier robots calculate their service load for the purpose of aiding their respective edge servers in matching a new task request to a carrier robot.

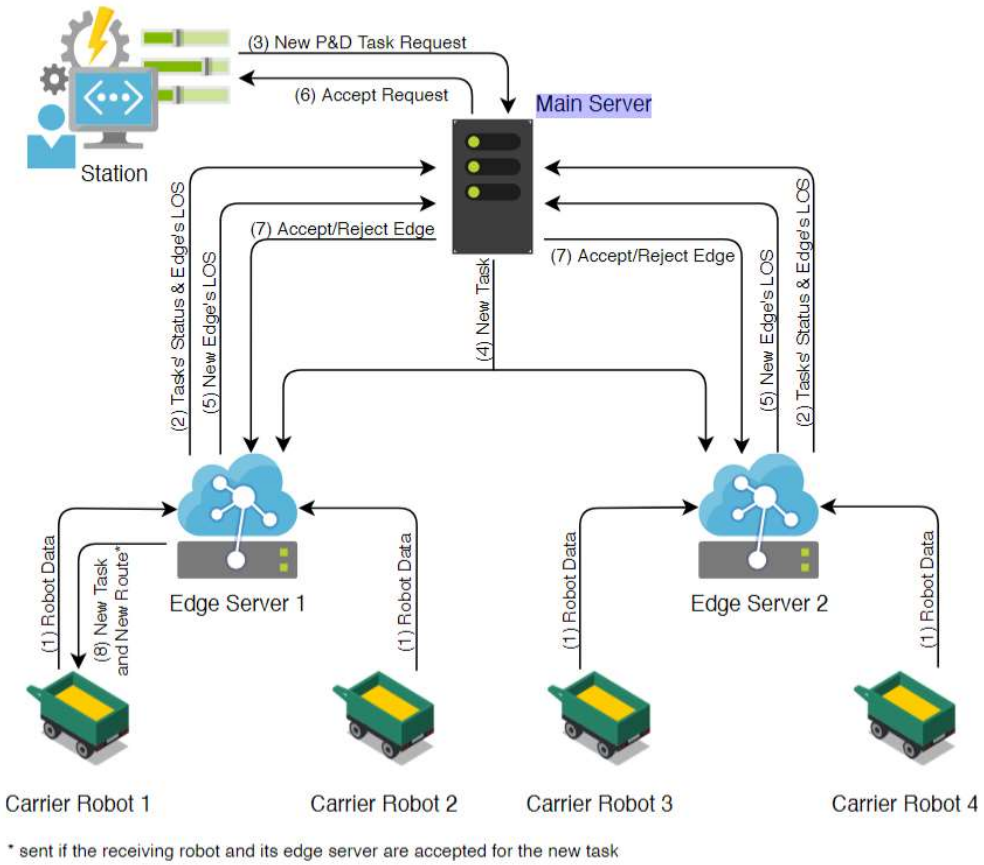

Figure 3. Flow of Messages in Proposed Framework

The carrier robot's service load is a numerical value with arbitrary units representing how much service the carrier robot does. It incorporates the carrier robot's average payload and detour ratio over its current travel route. Let $q_3 = \{q_{3,1}, q_{3,2}, \ldots, q_{3,Q_3}\} = q_1 \cup q_2$ be the full route of a robot, with $Q_3 = |q_3| = Q_1 + Q_2$. The average payload of a robot over the full route is formulated as follows:

$$L_{ave} = \begin{cases} \frac{1}{Q_3}\sum_{k=1}^{Q_3} L_{q_{3,k}}, & z = 2 \vee z = 3, \\ 0, & z = 1 \end{cases} \quad (3)$$

The detour ratio of a task is a ratio between two distances, the distance from the task's origin station to the task's destination station in the full route and the direct distance from the task's origin station to the task's destination station. In simple words, it is a ratio between the real traveled distance and the shortest distance to execute the task. The detour ratio of a task r is formulated as follows:

$$\Upsilon_r = \frac{d_{route}(o_r, d_r, Q_3)}{\mathfrak{D}(o_r, d_r)} \quad (4)$$

where $d_{route}(o_r, d_r, Q_3)$ is a function that returns the distance from or to d_r in Q_3. The carrier robot will then calculate the average detour ratios of currently assigned tasks as follows:

$$\widetilde{\Omega} = \gamma \frac{\mathrm{L_{ave}}}{\mathrm{L_{\theta}}} + (1-\gamma)\frac{\Upsilon_{\mathrm{ave}}}{\Upsilon_{\theta}} \tag{5}$$

The carrier robot then calculates its service load with the formula as follows:

$$\widetilde{\Omega} = \gamma \frac{\mathrm{L_{ave}}}{\mathrm{L_{\theta}}} + (1-\gamma)\frac{\Upsilon_{\mathrm{ave}}}{\Upsilon_{\theta}} \tag{6}$$

where γ acts a balancing variable for the average payload and average detour ratio. This service load value, along with other data of the carrier robot, is uploaded to its edge server for monitoring and controlling purposes.

The edge servers calculate their service load to help the main server decide which edge server is fit to be assigned to the new task request. Assume that $m_e = \{\mathrm{m_{e,1}}, \mathrm{m_{e,2}}, \dots, \mathrm{m_{e,M_e}}\}$ is a set of $\mathrm{M_e}$ carrier robots led by an edge server e in a factory. The edge server's service load can be determined by calculating the average of its robots' service loads as follows:

$$\widehat{\Omega_{\mathrm{e}}} = \frac{1}{M_{\mathrm{e}}} \sum_{m \in m_e} \widetilde{\Omega_m}. \tag{7}$$

Then, all edge servers will periodically upload their service load to the main server. When a new task request arrives at the main server, only N edge servers with the lowest service load will be selected for consideration. Each selected edge server then will receive the new task from the main server and run it using Task Insertion Algorithm to decide which carrier robot under its control is the best for the new task.

The Task Insertion Algorithm is designed based on Discrete Particle Swarm Optimization (PSO) Algorithm as the main part. Because the problem in (2) falls into a combinatorial problem, the PSO Algorithm is modified into Discrete PSO. The Task Insertion Algorithm is shown in Algorithm 1.

Line 1 gives the function name, and the inputs required. Line 2-14 gives initialization of the Task Insertion Algorithm, creating particles (solutions) population by randomly generating the position and velocity from a uniform distribution and creating variable to hold the new next-stations list $\widetilde{q_2}$. The main loop is shown in Line 15-52. Variable w is a parameter to control how far a particle can move from its current position. The value gets less gradually over time to make particles cover more coordinates in the search space, compared with a fixed value. Each particle is inspected inside the main loop against the objective function. The particles move in a two-dimensional search space with real number coordinate with a value range from 0 to 100. The position coordinates then are converted into discrete positive integer number with a value range from 0 to $\widetilde{Q_2}$, as shown in Line 18-19. Then the new coordinate is used to arrange $\widetilde{q_2}$'s sequence, as shown in Line 20-33. The arranged $\widetilde{\boldsymbol{q_2}}$ is then checked for objective function value using (3), as shown in Line 34-37. Then, the personal best of every particle and global best are updated, as shown in Line 38-46. At the end of the main loop, every particle's velocity and position are updated to be used when the loop restarts, as shown in Line 47- 49. When the main loop finishes looping, it will return the gained total distance of the new next-stations list $\mathrm{F_g}$ and the newly updated next-stations list $\widetilde{q_2^{best}}$. If Task Insertion Algorithm returns $\mathrm{F_g} = \infty$, it means the new task is unsuitable for insertion and thus will be rejected.

When all carrier robots of an edge server e have been checked for compatibility with the new task, it will then have a set of total distances $F_{g,m_e} = \{F_{g,m_{e,1}}, F_{g,m_{e,2}}, \dots, F_{g,m_{e,M_e}}\}$ and a set of the best next- stations lists of each edge server e's carrier robot $\widetilde{q_{2,e}^{best}} = \{\widetilde{q_{2,m_{e,1}}^{best}}, \widetilde{q_{2,m_{e,2}}^{best}}, \dots, \widetilde{q_{2,m_{e,M_e}}^{best}}\}$. The edge server e then will find the minimum total distance from the set as follows:

$$\mathrm{F_{g,m_e^{select}}} = \min\left(\mathrm{F_{g,m_e}}\right), \tag{8}$$

with the carrier robot $m_{\mathrm{e}}^{select} \in m_e$ and $\widetilde{q_{2,m_{\mathrm{e}}^{select}}^{best}} \in \widetilde{q_{2,e}^{best}}$ that are tied to the minimum total distance. The edge server will also recalculate its own and carrier robot m_{e}^{select}'s service load as additional data to be

used later by the main server. Then, these data are uploaded by each edge server (that can include the new task in its task queue) to the main server.

Algorithm 1. The Task Insertion Algorithm to Include New Task's Origin and Destination Station into Next-Stations List

function INSERTNEWTASK($o_{r_{\text{new}}}, d_{r_{\text{new}}}, \mathbf{r}, \mathbf{q_1}, \mathbf{q_2}$)
  # Initialize particles' position and velocity.
  for $i \leftarrow 1$, popSize **do**
    $\mathtt{x}_{i,1} \sim \mathcal{U}(0, 100)$ *# Representing $o_{r_{\text{new}}}$'s index.*
    $\mathtt{x}_{i,2} \sim \mathcal{U}(0, 100)$ *# Representing $d_{r_{\text{new}}}$'s index.*
    $\mathtt{X}_i \leftarrow [\mathtt{x}_{i,1}\ \mathtt{x}_{i,2}]$ *# Particle i's position in search space.*
    $\mathtt{F}_{\mathtt{X}_i} \leftarrow \infty$ *# Particle i's objective function value.*
    $\mathtt{p}_i \leftarrow \mathtt{X}_i$ *# Particle i's best position.*
    $\mathtt{F}_{\mathtt{p}_i} \leftarrow \infty$ *# Particle i's best objective function value.*
    $\mathtt{v}_{i,1} \sim \mathcal{U}(-50, 50)$
    $\mathtt{v}_{i,2} \sim \mathcal{U}(-50, 50)$
    $\mathtt{V}_i \leftarrow [\mathtt{v}_{i,1}\ \mathtt{v}_{i,2}]$ *# Particle i's velocity .*
  end for
  $\breve{Q}_2 \leftarrow |\mathbf{q_2}| + 2$
  $\breve{\mathbf{q}}_\mathbf{2} \leftarrow \left\{\breve{q}_{2,1}, \breve{q}_{2,2}, \ldots, \breve{q}_{2,\breve{Q}_2}\right\}$
  # Begin the main loop.
  for loop $\leftarrow 1$, maxLoop **do**
    $\mathtt{w} \leftarrow (\mathtt{maxLoop} - \mathtt{loop})/\mathtt{maxLoop}$
    for $i \leftarrow 1$, popSize **do**
    # Convert particle's position from real number to integer number.
      $x_1 \leftarrow \text{round}\left(\frac{\mathtt{x}_{i,1}}{100}\left(\breve{Q}_2 - 1\right)\right) + 1$
      $x_2 \leftarrow \text{round}\left(\frac{\mathtt{x}_{i,2}}{100}\left(\breve{Q}_2 - 1\right)\right) + 1$
    # Combine $o_{r_{\text{new}}}$ and $d_{r_{\text{new}}}$ into next-stations list.
      if $x_2 \leq x_1$ **then** *# $o_{r_{\text{new}}}$ must precede $d_{r_{\text{new}}}$.*
        $\mathtt{F}_{\mathtt{X}_i} \leftarrow \infty$
      else
        idx $\leftarrow 1$
        for idxx $\leftarrow 1$, $\breve{Q}_2$ **do**
          if idxx $= x_1$ **then**
            $\breve{q}_{2,\mathtt{idxx}} \leftarrow o_{r_{\text{new}}}$
          else if idxx $= x_2$ **then**
            $\breve{q}_{2,\mathtt{idxx}} \leftarrow d_{r_{\text{new}}}$
          else
            $\breve{q}_{2,\mathtt{idxx}} \leftarrow q_{2,\mathtt{idx}}$
            idx $\leftarrow$ idx $+ 1$
          end if
        end for
      # Find objective function value of particle i using (2).
        $\mathtt{F}_{\mathtt{X}_i} \leftarrow \mathfrak{D}(q_{1,Q_1}, \breve{q}_{2,1}) + \sum_{k=1}^{\breve{Q}_2 - 1} \mathfrak{D}(\breve{q}_{2,k}, \breve{q}_{2,k+1})$
        if any constraint is violated **then**
          $\mathtt{F}_{\mathtt{X}_i} \leftarrow \infty$
        end if
        if $\mathtt{F}_{\mathtt{X}_i} < \mathtt{F}_{\mathtt{p}_i}$ **then**
      # Update Particle i's best objective function value.
          $\mathtt{F}_{\mathtt{p}_i} \leftarrow \mathtt{F}_{\mathtt{X}_i}$
      # Update Particle i's best position.
          $\mathtt{p}_i \leftarrow \mathtt{X}_i$
          if $\mathtt{F}_{\mathtt{p}_i} < \mathtt{F}_\mathtt{g}$ **then**
        # Update global best objective function value.
            $\mathtt{F}_\mathtt{g} \leftarrow \mathtt{F}_{\mathtt{p}_i}$
        # Update global best position.
            $\mathtt{g} \leftarrow \mathtt{p}_i$
        # Update global best next-stations list.
            $\breve{\mathbf{q}}_\mathbf{2}^{\text{best}} \leftarrow \breve{\mathbf{q}}_\mathbf{2}$
          end if
        end if
      # Update Particle i's velocity.
        r1, r2 $\sim \mathcal{U}(0, 1)$
        $\mathtt{V}_i \leftarrow \mathtt{w}\,\mathtt{V}_i + \mathtt{c1}\,\mathtt{r1}\,(\mathtt{p}_i - \mathtt{X}_i) + \mathtt{c2}\,\mathtt{r2}\,(\mathtt{g} - \mathtt{X}_i)$
      # Update Particle i's position.
        $\mathtt{X}_i \leftarrow \mathtt{X}_i + \mathtt{V}_i$
      end if
    end for
  end for
  return $\mathtt{F}_\mathtt{g}$, $\breve{\mathbf{q}}_\mathbf{2}^{\text{best}}$
end function

Let $e = \{\mathrm{e}_1, \mathrm{e}_2, \ldots, \mathrm{e}_E\}$ be a set of E edge servers, $F_{g,e} = \left\{F_{g,m_{\mathrm{e}_1}^{select}}, F_{g,m_{\mathrm{e}_2}^{select}}, \ldots, F_{g,m_{\mathrm{e}_E}^{select}}\right\}$ be a set of total distances from every edge server, $q_{2,m_e^{select}}^{\widetilde{best}} = \left\{q_{2,m_{\mathrm{e}_1}^{select}}^{\widetilde{best}}, q_{2,m_{\mathrm{e}_2}^{select}}^{\widetilde{best}}, \ldots, q_{2,m_{\mathrm{e}_E}^{select}}^{\widetilde{best}}\right\}$ be a set of best next-stations list of each edge server, $\widehat{\mathfrak{L}_e} = \{\widehat{\mathfrak{L}_{\mathrm{e}_1}}, \widehat{\mathfrak{L}_{\mathrm{e}_1}}, \ldots, \widehat{\mathfrak{L}_{\mathrm{e}_E}}\}$ be a set of service load of every edge server, and $\mathfrak{L}_{m_e^{select}} = \left\{\mathfrak{L}_{m_{\mathrm{e}_1}^{select}}, \mathfrak{L}_{m_{\mathrm{e}_2}^{select}}, \ldots, \mathfrak{L}_{m_{\mathrm{e}_E}^{select}}\right\}$ be a set of service load of every selected carrier robot of every edge server. For example, if all carrier robots of edge server e_1 have been checked for compatibility with the new task, edge server e_1 will have $F_{g,m_{\mathrm{e}_1}}$.

The same will happen to other edge servers (edge server e_2 will have $F_{g,m_{\mathrm{e}_2}}$, and so on for all edge servers). Each edge servers will then choose the minimum value in their respective set using (8) (e.g., edge server e_1 will get $F_{g,m_{\mathrm{e}_1}^{select}}$. Then, all edge servers will upload their respective output to main server, forming set $F_{g,e}$, which will be input for Algorithm 2.

Algorithm 2. The Selection Algorithm in Main Server for Selecting an Edge Server and A Carrier Robot for The New Task Execution

```
1:  function SELECTEXECUTOR(F_{g,e}, q̆^{best}_{2,m^{select}_e}, Ω̂_e, Ω̆_{m^{select}_e})
2:      bestEdge ← e_1
3:      bestRobot ← m^{select}_{e_1}
4:      for j ← 2, E do
5:          if [F_{g,e_j} < F_{g,e_{j−1}}] or
6:          [F_{g,e_j} = F_{g,e_{j−1}} and Ω̂_{e_j} < Ω̂_{e_{j−1}}] or
7:          [F_{g,e_j} = F_{g,e_{j−1}} and Ω̂_{e_j} = Ω̂_{e_{j−1}} and Ω̆_{m^{select}_{e_j}} < Ω̆_{m^{select}_{e_{j−1}}}]
8:          then
9:              bestEdge ← e_j
10:             bestRobot ← m^{select}_{e_j}
11:             bestNextStaList ← q̆^{best}_{2,m^{select}_{e_j}}
12:         end if
13:     end for
14:     return bestEdge, bestRobot, bestNextStaList
15: end function
```

The main server then picks an edge server and a carrier robot to execute the task with algorithm shown in Algorithm 2. Three conditions for an edge and the best carrier robot to be selected is:

- if the total distance of the edge server is the least compared with other edge servers, or
- if there is another edge with the same total distance but the edge server's service load is the least com- pared with other edge servers, or
- if there is another edge with the same total distance and service load but the service load of the edge server's selected carrier robot is the least among other edge server's selected carrier robots,

which is shown in Line 5-11 of Algorithm 2. After the main server has selected the best edge server and the best carrier robot, the main server will send an approval message to the best edge server and disapproval messages to other edge servers considered in selection. The edge server that receives approval from main server will update its selected carrier robot data by sending the new task and new next-stations list. The selected carrier robot will update itself with data from its edge server and start executing its tasks with the new task already added.

4. Finding and Discussion

We performed discrete-event simulations to analyze how the proposed framework fares within the designated scenario. The simulations are designed and run-on MATLAB R2022a. The simulations were designed to output the proposed framework's performance within twelve hours of operation (43,200 s), with fixed-increment time progression (one second is represented by a time step). In each time-step, the probability of a new task request in each time-step is set to 10 %. The parameters for the simulation are shown in Table 1, and the floor layout of the factory is shown in Figure 4. For simulations, it is assumed that multiple charging platforms are available for carrier robots to use at the robot pool when they return, and carrier robots depart with full capacity and enough for executing tasks until return to robot pool again.

Table 1. Parameters in Simulation

Parameter	Value
Number of working stations S	10
Maximum pickup waiting duration T_{wait}	600 s
Maximum task delivery detour ratio Υ_θ	1.5
Carrier robot's speed while moving	2 m/s
Carrier robot's capacity L_θ	150 kg
Carrier robot's balancing variable γ	0.5
PSO's population size popSize	10
PSO's maximum iteration number maxLoop	100
PSO's acceleration factors $c1, c2$	2
Simulated duration	43,200 s
Probability of a new task request in each time step	10%
Factory area size	100 m × 100 m

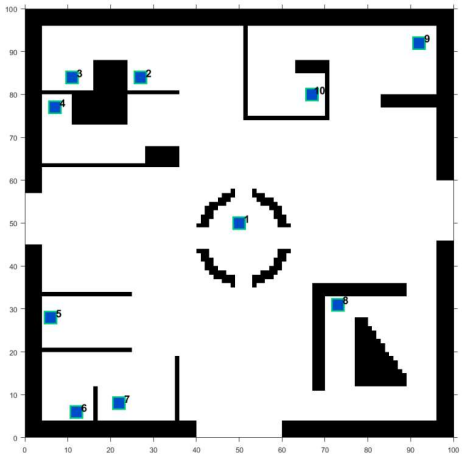

Figure 4. The Floor Layout of The Factory Simulation.
The Blue Squares Mark the Position of Working Stations in the Factory

The utilization of edge servers has proven to be beneficial in the proposed framework as an intermediary between the main server and carrier robots. Figure 5 and Figure 6 show the number of messages the main server received and sent during twelve hours of operation with and without edge servers' utilization.

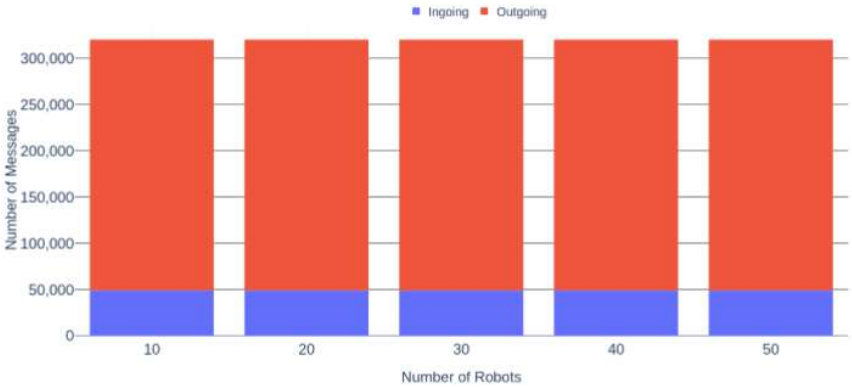

Figure 5. The Number of Messages the Main Server Received and Sent in Twelve Hours of Operation *with* Utilization of Edge Servers

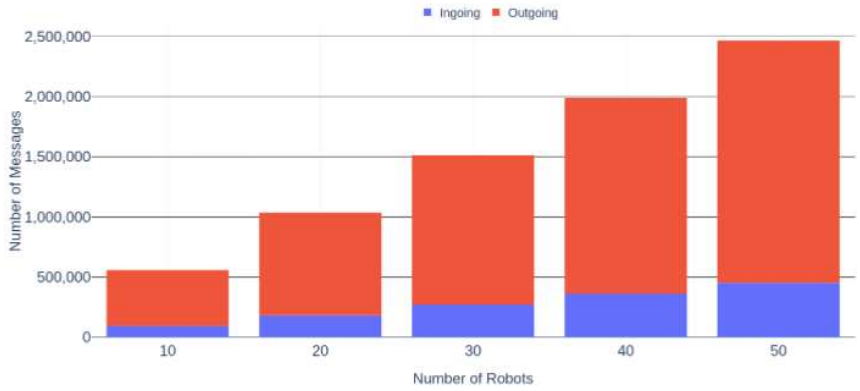

Figure 6. The Number of Messages the Main Server Received and Sent in Twelve Hours of Operation *without* Utilization of Edge Servers

When edge servers are utilized, a significant drop in the number of processed messages by the main server is attained. With edge servers, even when the number of carrier robots increases, the number of processed messages remains the same. Even the proposed framework with five edge servers and fifty carrier robots processed fewer messages than a framework with ten carrier robots but no edge servers.

The carrier robots employed have a significant impact on tasks' execution. Figure 7 shows that the number of tasks dropped by the proposed framework decreases as the number of robots increases.

Figure 7. The Number of Tasks That Have Been Delivered, Still in Progress, and Dropped in Twelve Hours of Operation with Utilization of Five Edge Servers and Varying Number of Carrier Robots

With twenty robots, the number of dropped tasks can be decreased to zero. With thirty robots, the number of tasks still in progress can be decreased to seven. The same number is produced by forty robots and fifty robots because the last seven tasks were still in progress and were requested near the end of the simulation and did not have enough time to be fully executed. Increasing the number of carrier robots also gives less time for each task to be fully executed. As shown in Figure 8, significant decrease in average task execution duration happens when robot numbers increase from ten to twenty. However, no significant change happens when the carrier robot number increases by more than twenty. A similar condition also happens with the average task delivery detour ratio. Task delivery detour ratio is a ratio between the actual distance taken to deliver tasks from its origin to the destination and the direct distance between that origin and destination. Task delivery detour ratio of one for a task means the task is delivered directly from its origin to destination. Figure 9 shows that the detour ratio can be decreased when the number of carrier robots increases from ten to twenty, but no significant changes when it is increased more.

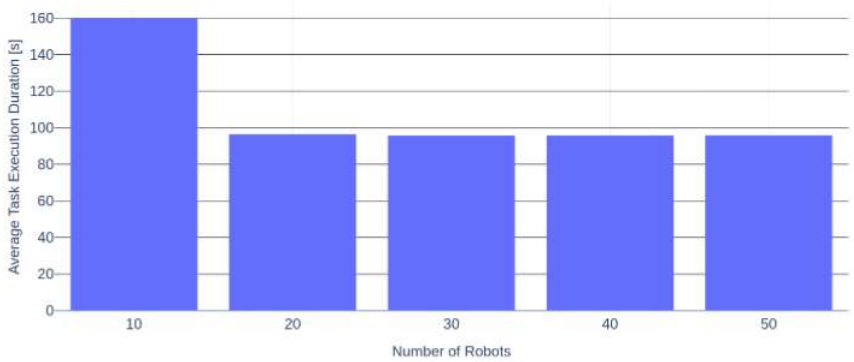

Figure 8. The average Task Execution Duration in Twelve Hours of Operation with Utilization of Five Edge Servers and Varying Number of Carrier Robots

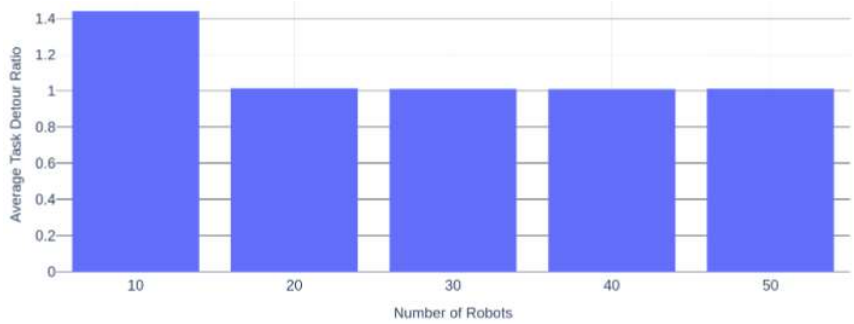

Figure 9. The Average Task Delivery Detour Ratio in Twelve Hours of Operation with Utilization of Five Edge Servers and Varying Number of Carrier Robots

Having an optimum number of carrier robots is necessary to maintain the balance between the service load of each robot and task execution performance. As shown in Figure 10 and Figure 11, the average number of tasks handled, and the average traveled distance per robot decrease as the number of robots is increased. Considering the previous results, having thirty robots can be taken as an optimum number, which can give excellent performance and even save the initial cost of procuring the robots and operational and maintenance cost for the robot because the number of tasks assigned, and distance traveled are reduced for each robot.

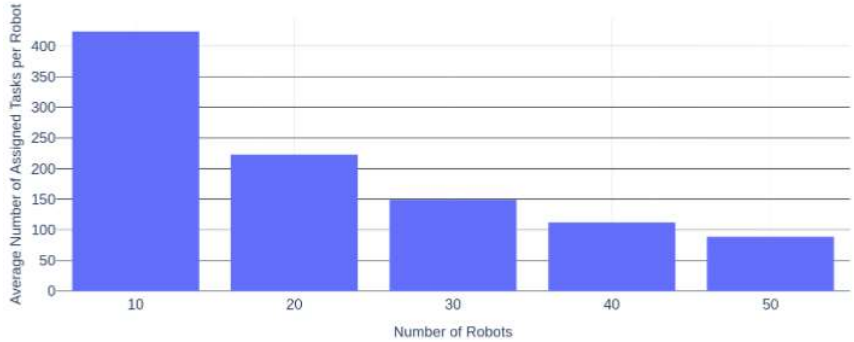

Figure 10. The Average Number of Tasks Assigned for Each Robot in Twelve Hours of Operation with Utilization of Edge Servers and Varying Number of Carrier Robots

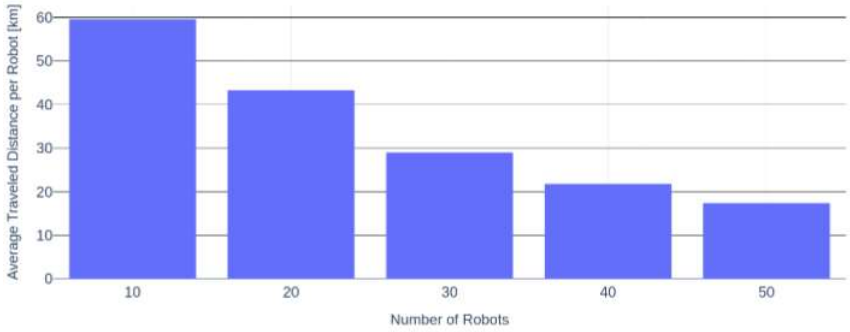

Figure 11. The Average Traveled Distance for Each Robot in Twelve Hours of Operation without Utilization of Edge Servers and Varying Number of Carrier Robots

5. Conclusion

This paper introduces a framework designed to optimize the pickup and delivery (P&D) tasks of carrier robots within a factory environment, utilizing edge servers to improve communication and task allocation. The discrete-event simulations indicate that integrating edge servers substantially alleviates the communication load on the main server while preserving task processing efficiency, irrespective of the number of carrier robots utilized. The results demonstrate a distinct correlation between the quantity of carrier robots and the efficacy of the P&D tasks. The increase in the number of robots results in a reduction of dropped tasks and enhances task execution efficiency. Results indicate that beyond a threshold of thirty robots may lead to diminishing returns regarding execution efficiency and delivery detour ratios. This configuration effectively balances the maximization of task handling capabilities with the minimization of operational costs for each robot. The proposed framework presents a feasible approach to improving the efficiency of P&D operations in smart factories, illustrating the transformative potential of edge computing in robotic logistics. Future research may investigate adaptive strategies for real-time task allocation to enhance performance across diverse operational conditions.

Acknowledgment

This work was supported by Internal Funding Research Program of Telkom University (KWR4.030/ LIT06/PPM-LIT/2024).

References

[1] A. Yildirim, H. Reefke, and E. Aktas, Mobile Robot Systems and Their Evaluation. Cham: Springer International Publishing, 2023, pp. 17–47.

[2] H. Ding, Y. Huang, J. Shi, Q. Shi, and Y. Yang, “A Novel Industrial AGV Control Strategy Based on Dual-Wheel Chassis Model,” Assembly Automation, vol. 42, no. 3, pp. 306–318, 2022.

[3] X. Liang, G. H. de Almeida Correia, K. An, and B. van Arem, “Automated Taxis’ Dial-a-Ride Problem with Ride-Sharing Considering Congestion-Based Dynamic Travel Times,” Transportation Research Part C: Emerging Technologies, vol. 112, pp. 260–281, 2020.

[4] K. Gkiotsalitis, “The Dial-a-Ride Problem Considering the in-Vehicle Crowding Inconvenience due to COVID-19,” in 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), 2021, pp. 3746–3751.

[5] M. Abedi, R. Chiong, R. Athauda, H. Seidgar, Z. Michalewicz, and A. Sturt, “A Regional Multi-Objective Tabu Search Algorithm for a Green Heterogeneous Dial- a-Ride Problem,” in 2019 IEEE Congress on Evolutionary Computation (CEC), 2019, pp. 2082–2089.

[6] A. Syed, I. Gaponova, and K. Bogenberger, “Neural Network-Based Metaheuristic Parameterization with Application to the Vehicle Matching Problem in Ride-Hailing Services,” Transportation Research Record, vol. 2673, no. 10, pp. 311–320, 2019.

[7] Q. Tang and M. G. Armellini, "An Ant Colony Algorithm with Penalties for the Dial-a- Ride Problem with Time Windows and Capacity Restriction," in 2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), 2021, pp. 1–6.

[8] M. S. Hossain, C. I. Nwakanma, J. M. Lee, and D.-S. Kim, "Edge Computational Task Offloading Scheme Using Reinforcement Learning for IIoT Scenario," ICT Express, vol. 6, no. 4, pp. 291–299, 2020. [Online]. Available: https://doi.org/10.1016/j.icte.2020.06.002. [Accessed: August, 2024].

[9] D. Guo, S. Gu, J. Xie, L. Luo, X. Luo, and Y. Chen, "A Mobile-Assisted Edge Computing Framework for Emerging IoT Applications," vol. 17, no. 4, Jul 2021.

[10] W. Liang, Y. Ma, W. Xu, Z. Xu, X. Jia, and W. Zhou, "Request Reliability Augmentation with Service Function Chain Requirements in Mobile Edge Computing," IEEE Transactions on Mobile Computing, vol. 21, no. 12, pp. 4541–4554, 2022.

[11] W. Dai, H. Nishi, V. Vyatkin, V. Huang, Y. Shi, and X. Guan, "Industrial Edge Computing: Enabling Embedded Intelligence," IEEE Industrial Electronics Magazine, vol. 13, no. 4, pp. 48–56, 2019.

[12] A. Rahman, J. Jin, A. Rahman, A. Cricenti, M. Afrin, and Y. Ning Dong, "Energy-Efficient Optimal Task Offloading in Cloud Networked Multi-Robot Systems," Computer Networks, vol. 160, pp. 11 – 32, 2019.

[13] L. Hu, Y. Miao, G. Wu, M. M. Hassan, and I. Humar, "iRobot-Factory: An Intelligent Robot Factory Based on Cognitive Manufacturing and Edge Computing," Future Generation Computer Systems, vol. 90, pp. 569 – 577, 2019.

[14] M. Afrin, J. Jin, A. Rahman, Y.-C. Tian, and A. Kulkarni, "Multi-Objective Resource Allocation for Edge Cloud Based Robotic Workflow in Smart Factory," Future Generation Computer Systems, vol. 97, pp. 119 – 130, 2019.

[15] Y. Lian, Q. Yang, W. Xie, and L. Zhang, "Cyber- Physical System-Based Heuristic Planning and Scheduling Method for Multiple Automatic Guided Vehicles in Logistics Systems," IEEE Transactions on Industrial Informatics, vol. 17, no. 11, pp. 7882–7893, 2021.

[16] Z. Yan, B. Ouyang, D. Li, H. Liu, and Y. Wang, "Network Intelligence Empowered Industrial Robot Control in the F-RAN Environment," IEEE Wireless Communications, vol. 27, no. 2, pp. 58–64, 2020.

[17] R. A. de Omena, D. F. Santos, A. Perkusich, and D. C. Valadares, "Two-tier MPC Architecture for AGVs Navigation Assisted by Edge Computing in An Industrial Scenario," Internet of Things, vol. 21, p. 100666, 2023.

[18] P. T. Daely, A. Putri Anantha, J. M. Lee, and D.-S. Kim, "Distributed Computing Architecture for Logistic Job Allocation in Smart Factory," in 2020 International Conference on Information and Communication Technology Convergence (ICTC), 2020.