GENERAL INSTRUCTION

- **Authors: When you submit your corrections, please either annotate the IEEE Proof PDF or send a list of corrections. Do not send new source files as we do not reconvert them at this production stage.**
- **Authors: Carefully check the page proofs (and coordinate with all authors); additional changes or updates WILL NOT be accepted after the article is published online/print in its final form. Please check author names and affiliations, funding, as well as the overall article for any errors prior to sending in your author proof corrections. Your article has been peer reviewed, accepted as final, and sent in to IEEE. No text changes have been made to the main part of the article as dictated by the editorial level of service for your publication.**
- **Authors: Per IEEE policy, one complimentary proof will be sent to only the Corresponding Author. The Corresponding Author is responsible for uploading one set of corrected proofs to the IEEE Author Gateway.**

QUERIES

Q1. Please provide ORCID for the author Luís A. Alexandre in the byline.
Q2. Please confirm or add details for any funding or financial support for the research of this article.
Q3. Please check and confirm whether the author affiliations in the first footnote are correct as set.
Q4. Please provide complete bibliography details for Refs. [25] and [35].

Static Algorithm Allocation With Duplication in Robotic Network Cloud Systems

Saeid Alirezazadeh and Luís A. Alexandre

***Abstract*—Robotic networks can be used to accomplish tasks that exceed the capacity of a single robot. In a robotic network, robots can work together to accomplish a common task. Cloud robotics allows robots to benefit from the massive storage and computing power of the cloud. Previous studies mainly focus on minimizing the cost of resource retrieval by robots by knowing the resource allocation in advance. Duplicating algorithms on multiple nodes can reduce the total time required to execute a task. We address the question of which algorithms should be duplicated and where the duplicates should be placed to improve overall performance. We have developed a procedure to answer wherein a robotic network cloud system should algorithms be executed and whether they should be duplicated to achieve optimal performance in terms of overall task execution time for all robots. Our proposed duplication procedure is optimal in the sense that the number of duplicated algorithms is minimal, while the result provides minimal overall completion time for all robots.**

***Index Terms*—Human-robot collaboration, job completion time, monitoring, quality metric, task scheduling.**

I. Introduction

The use of robots is rapidly increasing in various areas of human life, e.g., domestic [1], [2], [3], industrial and manufacturing [4], [5], [6], military [7], [8], [9], and others [10], [11].

To overcome the limitations of a single robot's capabilities, one can use multiple robots working together to complete a task. For example, lifting a heavy object may exceed the capacity of a single robot. Such a system of cooperative robots is called a robotic network.

The capacity of a robotic network is higher than that of a single robot, but the collective capacity of all robots limits the capacity of the robotic network [12]. Increasing the number of robots to increase the capacity could be the first solution to this limitation. However, increasing the number of robots increases the complexity of the model and the cost of the system. On the other hand, most of the tasks related to human-robot interaction, such as object, face, and speech recognition, are computationally intensive. Cloud robotics is a way to overcome the computational and capacity limitations of robots. It uses the Internet and cloud infrastructure to assign computations and share Big Data in real-time [13]. To achieve the optimal performance of cloud-based robotic systems, we need to solve the allocation problem. This is the problem that deals with deciding whether a newly arrived task should be uploaded to the cloud, executed on one of the robots (edge computing [14]), or processed on a server (fog computing [15]). The execution of tasks by a cloud robotic system is made possible by executing, collecting, and combining the results of several elementary tasks called algorithms. Before a robot executes a task, all the algorithms required for the task should be available and assigned to at least one of the processing units of the system. When a robot is assigned a task, the robot requests the outputs of the algorithms corresponding to the task. Our goal is to determine where the algorithms should be assigned to such that, regardless of which robot is performing the task, it retrieves all the required outputs of the algorithms in the shortest possible time.

The article is organized as follows. Section II reviews related work on task allocation and scheduling in robotic network cloud systems. Section III introduces some basic concepts that are central to this article. This is followed in Section IV by two procedures for identifying duplication algorithms. Section V describes the experimental methodology and discusses the results of the experiments[1]. And finally, Section VI draws some conclusions and points out future lines of work.

II. Related Work

Let a robotic network cloud system be capable of performing a finite set of tasks, T. Suppose that $\{A_1, \ldots, A_m\}$ is the set of all algorithms necessary to execute all tasks in T. Consider the case where the system is currently executing a subset of tasks, say T_1, when a new set of tasks, T_2, arrives. As we can see in Fig. 1, there are two types of task allocation:

- We refer to this sort of task allocation as static task allocation if we allocate the set of all algorithms in an effort to find the system's optimal performance.

Manuscript received 6 July 2022; revised 24 February 2023; accepted 12 April 2023. This work was supported in part by operation Centro-01-0145-FEDER-000019 - C4 - Centro de Competências em Cloud Computing, cofinanced by the European Regional Development Fund (ERDF) through the Programa Operacional Regional do Centro (Centro 2020), in the scope of the Sistema de Apoio à Investigação Científica e Tecnológica - Programas Integrados de IC&DT, in part by NOVA LINCS under Grant UIDB/04516/2020 with the financial support of FCT-Fundação para a Ciência e a Tecnologia, through national funds. Recommended for acceptance by L. Y. Chen. *(Corresponding*
Q2 *author: Saeid Alirezazadeh.)*

Saeid Alirezazadeh is with the C4-Cloud Computing Competence Center,
Q3 Universidade da Beira Interior, 6200-284 Covilhã, Portugal, and also with the Physikalische und Theoretische Chemie, Universität Graz, 8010 Graz, Austria (e-mail: saeid.alirezazadeh@gmail.com).

Luís A. Alexandre is with the NOVA LINCS, Universidade da Beira Interior, 6200-284 Covilhã, Portugal (e-mail: luis.alexandre@ubi.pt).

Digital Object Identifier 10.1109/TPDS.2023.3267293

[1]The code is available at https://github.com/SaeidZadeh/AlgorithmDuplication.

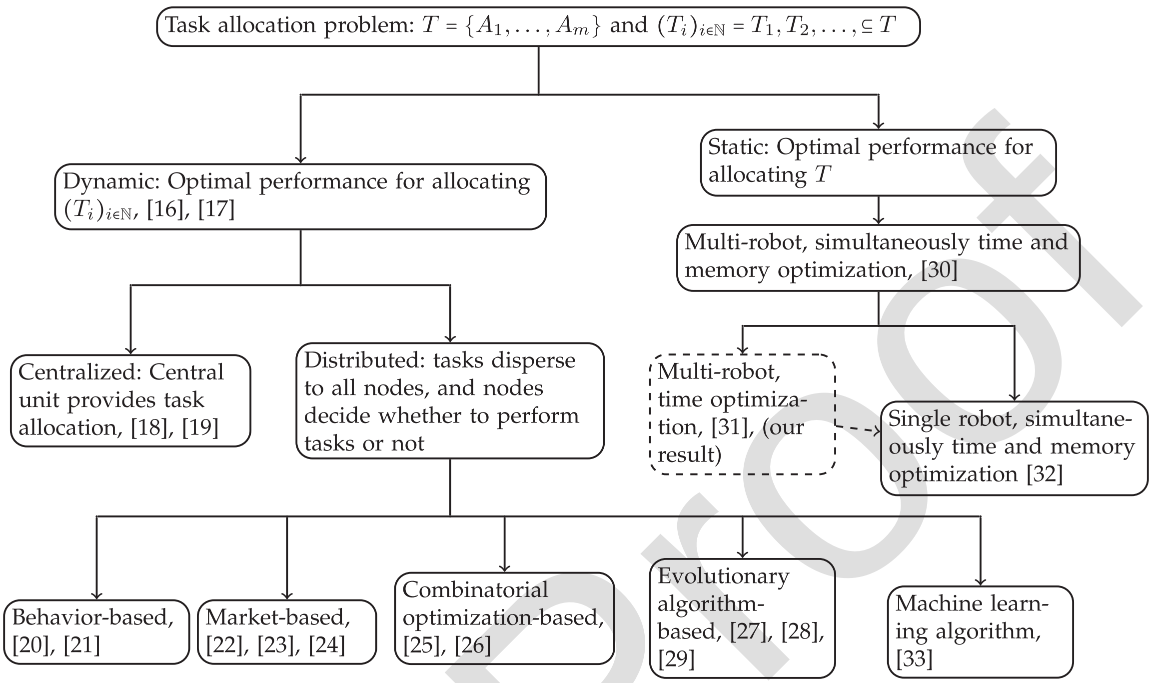


Fig. 1. Task allocation problem studied in literature. Algorithm i is represented by A_i. We used dashed arrow to show that the result in [32] cannot be extracted from the result of [31]. The dashed rectangle is used to show where our proposed stands, which is infact a procedure to be applied on the class of static allocation for multi-robots and optimizing the time.

- We refer to this sort of task allocation as dynamic task allocation if we look for the best performance by dynamically allocating the newly received work at various time steps.

[17] considered time window constraints for all tasks and proposed a load balancing procedure for cloud robotic systems; and [16] used a geometric approach to find an optimal task allocation by translating the allocation problem to a subspace of a hyperspace; [19] proposed Tercio, a centralized task allocation method that minimizes latency and physical proximity to tasks; [21] proposed aprocedure that considers the characteristics of cloud robotic architecture for optimal task assignment; [27] used evolutionary operators to find optimal task allocation; [26] studied resource sharing and presented a strategy to manage resources in near real-time; and [33] developed a scheduling technique to decrease response and makespan times and enhance resource effectiveness. Reinforcement learning is the foundation of the scheduling technique. They applied Bayes' theorem, the total task length of virtual machines at each time step is treated as independent, and the Q-values are estimated.

As shown in Fig. 1, the problems we address in this article focus on the static allocation problem. For more details on other works on dynamic task allocation mentioned in Fig. 1, see [32]. From the set of all algorithms, we can define a directed acyclic graph (DAG) whose vertices are the algorithms, and each edge (A_i, A_j) means that to execute A_j, the result of A_i is required. The graph of all algorithms shows the execution flows of all algorithms to accomplish a task.

In dynamic task allocation, we need to decide which task to assign to a node of the cloud robotic system after a set of tasks enters the system. When a task is assigned to a node, that node can ask other nodes to perform the necessary algorithms to accomplish the task. Occasionally, it is better to run these algorithms on nodes other than the one to which the task has been assigned in order to reduce memory usage and the time required to perform the task.

If the goal is to complete all possible tasks that the system can handle once, the most important aspect of a cloud robotic system will be the static task allocation. The goal of static allocation is to find the best way to distribute all algorithms required to execute all tasks in a way that minimizes the cost of task execution. The question of how to optimally distribute all algorithms among nodes is solved by static allocation. It achieves this by ensuring that the node to which the task is assigned optimally collects all the required data. In static task allocation, we reduce the cost of each task, regardless of where it is assigned. Static task allocation is as crucial as dynamic task allocation. It also demonstrates how to get cloud robotic systems to perform each task in the best possible way.

An algorithm assigned to multiple processors is called a duplicated algorithm. Because algorithms are interdependent, the output of a duplicated algorithm is more readily available to other processors to which its successor algorithms are assigned to. The following example is used to better explain the importance of algorithm duplication.

Example II.1. Let the architecture of the cloud robotic be as shown in Fig. 2. Suppose we have a task that requires only the output of a single algorithm. Given the output of the algorithm, the task can be performed by any of the edge nodes. Assume that the algorithm can be executed on each of the edge nodes, the fog node, and the cloud node with an average execution time of 3, 0.5, and 0.1 seconds, respectively. If we do not consider the duplication of algorithms, since we do not know

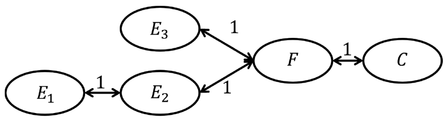


Fig. 2. Architecture of the robotic network cloud system with an average communication time between directed nodes of 1 seconds. E_i's are edge nodes for $i = 1, 2, 3$, F is the fog, and C is the cloud. The values on the edges are the average communication time.

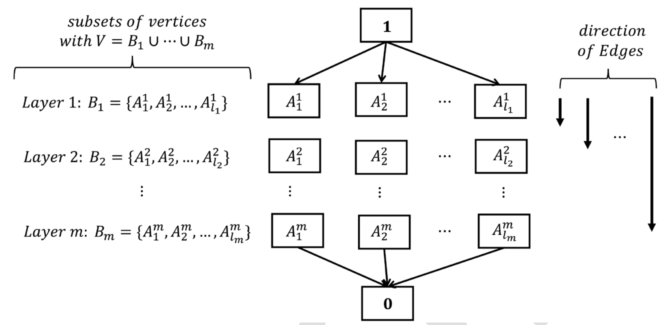


Fig. 3. Graph with downward edges. We add the virtual vertices $\mathbf{0}$ and $\mathbf{1}$ to the graph which creates a semi-lattice. Note that the l_i's for $i = 1, \ldots, m$ are not necessarily equal.

which of the edge nodes initiates the request to perform the task, the optimal performance of the system can be achieved by static algorithm allocation [30]. The result is the allocation of the algorithm to the fog node, where the average completion time of the task by the edge nodes E_1, E_2, and E_3 is 4.5, 2.5, and 2.5 seconds, respectively. If we are allowed to duplicate the algorithm, duplicating the algorithm on the edge node E_1 reduces the average task completion time by the edge node E_1 to 3 seconds. Thus, the optimal solution with minimum overal time to complete the task is to allocate the algorithm to the fog node F and duplicate it once on the edge node E_1.

The works [32] and [31] deal with static allocation but do not consider duplication of algorithms, which can improve optimal performance. To improve performance, [34] proposed a procedure for algorithm allocation with possible duplication. They provide a result that contains necessary (but not sufficient) conditions for task duplicability. They found that for a graph of all algorithms, duplicating an algorithm improves performance if the number of children of that algorithm or the number of children of at least one of its descendants is greater than or equal to 2. Their results reduce the space of algorithms whose duplications can improve performance. However, it is not specified exactly which algorithms need to be duplicated.

Our main goal is to define a procedure to find out which algorithms should be duplicated and where to allocate their duplicates to improve the performance of the system. We have proposed a recursive algorithm to determine which algorithms need to be duplicated, and where they should be allocated to, to improve the overall completion time.

III. Preliminaries

Before descrbing the procedure we recall several concepts from [30] that will be used to describe the main procedure.

Definition III.1. We construct a directed acyclic graph $G = (V, \overrightarrow{E})$ which helps us formulate a general model. The directed acyclic graph $G = (V, \overrightarrow{E})$ is defined by the set of algorithms $V = \{A_1, \ldots, A_m\}$ as the set of vertices of the graph G and $\overrightarrow{E}$ is a subset of the ordered pairs of elements of V

$$\overrightarrow{E} = \{(A_i, A_j) \mid A_j \text{ uses the result of } A_i\}.$$

Definition III.2. For a directed graph $G = (V, \overrightarrow{E})$ and $v \in V$, define:

- the number of elements of $\overrightarrow{E}$ in which v is the first component is called the out-degree of v

$$\text{OutDegree}(v) = |\{w \in V \mid (v, w) \in \overrightarrow{E}\}|.$$

- the number of elements of $\overrightarrow{E}$ in which v is the second component is called the in-degree of v

$$\text{InDegree}(v) = |\{w \in V \mid (w, v) \in \overrightarrow{E}\}|.$$

Remark 1. Some of the vertices of the directed graph in Definition III.1 must have in-degree 0, and some others must have out-degree 0.

By Remark 1, the graph can be represented by making sure that all of its edges point downward. The graph's vertices are displayed in various layers. All of the vertices in the first layer have in-degree zero. The second layer is made up of all the vertices with only edges in the graph G connecting them to the previous layer's vertices. And the following layer is made up of all the vertices with only edges in the graph G connecting them to the vertices of layers prior. As seen in Fig. 3, it is obvious that the last layer consists of all the vertices with an out-degree of zero. It is possible to think of the built-in graph with downward edges as a union of its connected components. In addition, add virtual vertices $\mathbf{0}$ and $\mathbf{1}$ to each of the connected graph components, with vertex $\mathbf{1}$ at the top of the first layer and edges drawn to all of the first layer's vertices, and the vertex $\mathbf{0}$ is at the bottom of the last layer and edges are drawn to it from all vertices of the last layer. This procedure turns the graph into a union of semi-lattices, $\mathcal{SL}(G)$. We slightly abuse the notation by using the symbols $\mathbf{0}$ and $\mathbf{1}$ to represent all the virtual vertices of all connected components of the graph. For more details, see [32].

Before discussing the procedure, we will modify the optimization problem so that its solution provides a solution to the static algorithm allocation. We follow a similar notation to the problem formulation in [31]:

- t_i^{st} is the time at which algorithm i was started (start time);
- t_i^{res} is the time when the execution of algorithm i finished (response time);
- V_n is the set of nodes in the cloud robotics architecture, including all nodes in the edge, nodes in the cloud (and nodes in the fog, but this is not assumed in [31]);
- x_{ik} indicates whether node k in V_n is assigned an algorithm i;
- the set of nodes E, F, and C subsets of V_n are used to indicate the set of edge nodes, fog nodes, or cloud nodes, respectively;

- V_t is the set of all algorithms in the graph of dependency of algorithms with additional virtual nodes;
- Z_{ik} is an indicator that an algorithm i can be assigned to a node k in V_n that can hold prior information about the places where algorithms need to be executed;
- pred_i is the set of algorithms that must be executed before algorithm i is executed (the set of all algorithms in the graph for which algorithm i is their successor);
- $(k, l) \in E_p$ indicates that nodes k and l are neighbors in the cloud robotics architecture, and E_p is the set of all neighbors;
- S_{ji} is the size of the intermediate data obtained from algorithm j that needs to be transmitted to the node executing algorithm i to be used as input for i;
- y_{ij} equals 1 if the task $i \in V_t$ is to be executed after $j \in V_t$, and 0 otherwise;
- R_{ik} is the runtime of algorithm i on node k.

The problem is to minimize the time between the initial request of the edge node e and the return of the last algorithm indexed by m (the algorithm **0**) to the edge node e (minimizing $t_m^{res}(= t_m^{res}(e))$), under the condition that each algorithm can only be executed by exactly one of the nodes, i.e.,

$$\sum_{k \in V_n} x_{ik} = 1\,, \quad \forall i \in V_t.$$

The prior knowledge of where to run the algorithms can be specified as follows:

$$\begin{array}{ll} x_{ik} \leq Z_{ik}, & \forall i \in V_t,\, k \in V_n \\ \sum_{k \in V_n} Z_{ik} \leq |E| + |F| + |C|, & \forall i \in V_t \\ \sum_{k \in V_n} Z_{ik} \geq 1, & \forall i \in V_t \\ Z_{ik} \in \{0, 1\}, & \forall i \in V_t,\, k \in V_n \end{array}$$

where Z_{ik}, takes values 0 and 1, is an indicator of whether an algorithm i can be assigned to a set of nodes to which k belongs and cannot be assigned to the remaining nodes. Note that we assume that $t_m^{st}(e) = 0$ for all $e \in E$, which means that the whole process of time minimization starts at time 0.

The time at which the algorithm i is started is the sum of the following times:

- the time to execute the set of all immediate predecessors of algorithm i

$$T_{1,i} = \max_{j \in \text{pred}_i} \left\{ t_j^{res} \left(\sum_{p \in V_n} x_{jp} p \right) \right\};$$

- the time taken to send intermediate data, used as input by algorithm i from the nodes generating these inputs to the node executing i

$$T_{2,i}^k = \sum_{j \in \text{pred}_i} \text{TransmissionTime}_k(S_{ji}),$$

  where $\text{TransmissionTime}_k(S_{ji})$ is the average time to transmit S_{ji} data to node k. From now on, we denote by $S_{i,j}$ all the additional information that needs to be transmitted to i in addition to the information obtained from the previous step to be used as input.

Consequently

$$t_i^{st} = T_{1,i} + \sum_{k \in V_n} x_{ik} T_{2,i}^k.$$

The time of termination of the algorithm i is the sum of the following times:

- the time at which algorithm i is started, t_i^{st};
- the runtime of algorithm i on node k;
- the average time to transmit the output data of algorithm i to the requested node p (the size of the output data of algorithm i is denoted by OutputSize_i).

Consequently

$$t_i^{res}(p) = t_i^{st} + \sum_{k \in V_n} x_{ik} R_{ik} + K,$$

where K is the average transmission time required by node p to obtain the required inputs of size OutputSize_i

$$K = \text{TransmissionTime}_p(\text{OutputSize}_i).$$

The preceding considerations imply the following formulation for minimizing time

$$\begin{aligned} &\min:\ t_m^{res} = \sqrt{\sum_{e=1}^{|E|} (t_m^{res}(e))^2} \\ &\text{s.t.}:\ \sum_{k \in V_n} x_{ik} = 1 \\ &\quad x_{ik} \leq Z_{ik},\ \forall i \in V_t,\, k \in V_n \\ &\quad 1 \leq \sum_{k \in V_n} Z_{ik} \leq |E| + |C| + |F|,\ \forall i \in V_t \\ &\quad t_i^{st} = T_{1,i} + \sum_{k \in V_n} x_{ik} T_{2,i}^k,\ \forall i \in V_t \\ &\quad T_{1,i} = \max_{j \in \text{pred}_i} \left\{ t_j^{res} \left(\sum_{p \in V_n} x_{jp} p \right) \right\},\ \forall i \in V_t \\ &\quad T_{2,i}^k = \sum_{j \in \text{pred}_i} \text{TransmissionTime}_k(S_{ji}), \\ &\qquad \forall i \in V_t,\, k \in V_n \\ &\quad t_i^{st} \geq \max_{j \in V_t} \left\{ \sum_{k \in V_n} x_{ik} x_{jk} y_{ij} \times \left(t_j^{res} \left(\sum_{p \in V_n} x_{jp} p \right) \right) \right\}, \\ &\qquad \forall i \in V_t \\ &\quad t_i^{res}(p) = t_i^{st} + \sum_{k \in V_n} x_{ik} R_{ik} \\ &\qquad + \text{TransmissionTime}_p(\text{OutputSize}_i),\ \forall i \in V_t \qquad (1) \\ &\quad x_{ik}, y_{ij}, Z_{ik} \in \{0, 1\}, \end{aligned}$$

where $t_j^{res}(e)$ is the response time of algorithm j when initiated by the edge node e. In this formulation, when a request for an algorithm A is sent by a node, the necessary algorithms $B_1, \ldots, B_q$ for executing algorithm A are requested from the nodes to which they are assigned, and then

- if the necessary conditions for the execution of algorithm B_i are satisfied, then algorithm B_i is executed, and its results are returned to the node that requested it;
- if the necessary conditions for the execution of algorithm B_i are not satisfied, then an iteration is performed in a similar way (requests for necessary algorithms are sent from the node to which algorithm B_i is assigned and on which it cannot be executed to the nodes to which the necessary algorithms for the execution of B_i are assigned).

Remark 2. In the optimization problem 1, an edge node requests the execution of an algorithm, and after the algorithm's corresponding node completes the request, the output is sent back to the original edge node. We suppose that every edge node can issue requests for any algorithm to be executed. When the initial edge node is altered, the response time of the final algorithm varies as a result of the architecture and neighborhood relationships between the nodes. The overall amount of time it takes for each edge node to obtain the final algorithm m result must be kept as low as possible. We must take into account the fact that each edge node has the ability to transmit requests for algorithms in order to determine how to distribute algorithms among nodes in a way that maximizes system performance as a whole. The virtual algorithm **0** is the final algorithm, thus in order to reduce the time, we suppose that each edge node sends a request for it. Depending on how the algorithms are distributed among the edge nodes, the final algorithm's response time varies in $\mathbb{R}^+$. In E-dimensional space, $\mathbb{R}^E$, the optimal allocation can be determined by minimizing the difference between the final algorithm's response times and the allocation of the algorithms for all the edge nodes, i.e., by minimizing

$$\sqrt{\sum_{r=1}^{|E|}(t_m^{res}(e))^2}.$$

One of the most appropriate methods to find an optimal solution to the problem 1 is to use the branch-and-bound method [35], which is described in [31].

IV. Procedures Identifying Algorithms for Duplication

We propose two procedures for identifying which algorithms should be duplicated. The first is based on combinatorial graph theory, i.e., algorithms of the same class[2] are duplicated and assigned to other nodes based on some constraints that ensure that duplication improves performance. The second proposal is a variation of the first procedure, where we solve optimization problems where the main objective is the overall time in which an edge node receives all the outputs of all the algorithms, and this is done for each edge node.

[2]Algorithms assigned to the same node in terms of the optimal solution without duplication.

A. Combinatorial Graph Theory

Assume that the set of all algorithms $A = \{A_1, \ldots, A_n\}$, the graph of all algorithms, G, and its respective semi-lattice $\mathcal{SL}(G)$ are known, and the robotic network cloud system is of a given architecture with edge nodes $\{E_1, \ldots, E_m\}$. For more details, see [32].

By the result 1 proposed for static algorithm allocation, we can find a solution to the allocation problem without algorithm duplication that minimizes

$$t_n^{res} = \sqrt{\sum_{i=1}^{m}(t_n^{res}(E_i))^2},$$

where $t_n^{res}(E_i)$ is the response time[3] for the node E_i. We then find the set of all execution flows, ExecutionFlows(G). Note that the value of $t_n^{res}(E_i)$ is equal to the maximum overall time of the elements of the set ExecutionFlows(G). We call the execution flow(s) with the maximum overall time for edge node E_i, the critical path(s) for edge node E_i. Also, note that duplicating a single algorithm can only improve $t_n^{res}(E_i)$ if and only if it improves the overall time of the critical path of E_i or reduces the number of critical paths in case there are more than one critical paths. Finally, note that we can improve t_n^{res} if and only if we improve at least one of $t_n^{res}(E_i)$, $i = 1, \ldots, m$.

Suppose that, for a given $\mathcal{SL}(G)$ and the architecture of the robotic network cloud system, we have obtained the optimal solution for the algorithm allocation.

An elementary step, denoted by Elementary$_{E_i}(A_j)$, is to duplicate the algorithm A_j in a critical path of the edge node E_i so that it improves the value of $t_n^{res}(E_i)$. Moreover, we define the stop step of the edge node E_i as follows: for any algorithm A_j in the critical path of E_i, the elementary step Elementary$_{E_i}(A_j)$ does not exist, i.e., the edge node E_i is in the stop step if the execution time of the task initiated by E_i cannot be improved. Note that if we are in the stop step of the edge node E_i, duplicating any algorithm that improves $t_n^{res}(E_k)$ for $E_k \neq R_i$ does not change $t_n^{res}(E_i)$, i.e., $t_n^{res}(E_i)$ is preserved by any duplication of any algorithm. In this case, $t_n^{res}(E_i)$ reaches its minimum possible value.

The goal is to execute elementary steps until all edge nodes reach their stop step.

Note that algorithms in a critical path are serial, and the Elementary$_{E_i}(A_j)$ can be made if and only if the minimum value of the sum of the following values:

- the communication time from the nodes where the immediate ancestors of algorithm A_j, are assigned to the new node to which we want to assign the copy of A_j;
- the communication time from the new node to which we want to assign the copy of A_j, to the nodes executing the immediate successor algorithms of A_j;
- the average execution time of algorithm A_j, on the new node we want to assign a copy of A_j;

[3]The response time of the node E_i means that we start at time 0 and find the time in which the execution of the algorithm **0** completes is equal to the maximum time required by the edge node E_i to obtain all outputs of all algorithms.

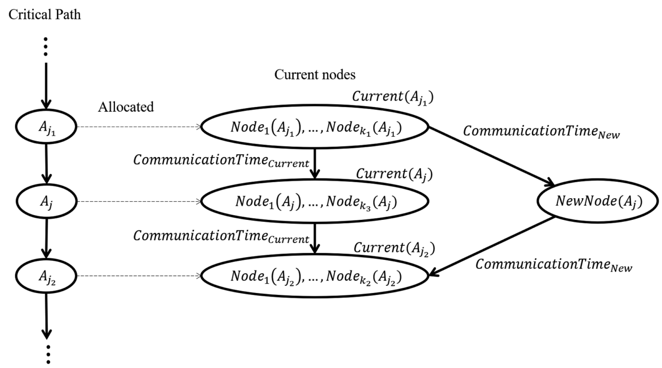


Fig. 4. Overview of an elementary step.

is less than the minimum value of the sum of the following values:

- the communication time from the nodes where the immediate ancestors of algorithm A_j are assigned, to the node executing algorithm A_j;
- the communication time from the new node of algorithm A_j, to the nodes that are its immediate successors;
- the average execution time of algorithm A_j, on the node on which it is assigned.

The elementary step for the edge node E_i and algorithm A_j, $\text{Elementary}_{E_i}(A_j)$, is shown in Fig. 4: Assume that a critical path of edge node E_i is given as a sequence of algorithms

$$\{\ldots, A_{j_1}, A_j, A_{j_2}, \ldots\}.$$

The elementary step is duplicating algorithm A_j to a new node such that the following holds

$$Ct_1 + Ct_2 + Ext > \min\{NCt_1 + NCt_2 + NExt\},$$

where

$$Ct_1 = \text{CommunicationTime}(Current(A_{j_1}), Current(A_j)),$$

$$Ct_2 = \text{CommunicationTime}(Current(A_j), Current(A_{j_2})),$$

$$NCt_1 = \text{CommunicationTime}(Current(A_{j_1}), NewNode(A_j)),$$

$$NCt_2 = \text{CommunicationTime}(NewNode(A_j), Current(A_{j_2})),$$

$$Ext = \text{ExecutionTime}_{A_j}(Current(A_j)),$$

and

$$NExt = \text{ExecutionTime}_{A_j}(NewNode(A_j)).$$

The level of improvement, which represents the saved overall time with this node addition, can be calculated as follows:

$$LevelofImproment(E_i, A_j, NewNode(A_j))$$
$$= Ct_1 + Ct_2 + Ext - \min\{NCt_1 + NCt_2 + NExt\}. \quad (2)$$

We will find the algorithm and the location of its duplication such that it maximizes the term (2) within all algorithms duplicated on a new node. Thus, duplicating this algorithm yields the most significant improvement in the value of $t_n^{res}(E_i)$ compared to other algorithms and their duplications.

Let $E_i \in E$ be the edge node requesting the outputs of all algorithms, $A_j \in V_t$ be an algorithm on a critical path, and $Current(A_j)$ be the node currently assigned to algorithm A_j. The main objective becomes

$$\begin{array}{rl} \max: & LevelofImproment(E_i, A_j, X) \\ \hline \text{s.t.}: & X \in V_n \end{array}$$

V_n is the set of nodes in the cloud robotics architecture. (3)

In the optimization (3), we will find the values of $LevelofImproment$ for all nodes.

- It will be negative when the overall time of the critical path is increased.
- It will be positive if the overall time of the critical path is decreased.
- It will be 0 if the overall time of the critical path does not change.

The latter is the case when the algorithm is assigned to the same node to which it is initially assigned to. Thus, the stop step is when the maximum value of $LevelofImproment$ within all X is equal to 0. So duplicating algorithms to other nodes will not improve the value of $LevelofImproment$ and the maximum value of $LevelofImproment$ for all nodes will be 0.

Note that algorithms in a critical path assigned to the same node should be duplicated simultaneously on the same node due to communication time. In other words, algorithms assigned to the same node will be considered as a single algorithm, and an elementary step will be applied to all of them simultaneously. For example, suppose that the sequence of algorithms $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ is a critical path and algorithms $\{A_1, A_3, A_4, A_6\}$ are assigned to the same node N_1 and algorithms $\{A_2, A_5\}$ are assigned to a different node N_2. Then, duplicating A_1 to node N_3 leads to duplicating $\{A_3, A_4, A_6\}$ to the same node N_3.

The procedure for improving t_n^{res} by duplicating algorithms is as follows: First, the solution of the optimal algorithm allocation problem is found without duplication. Then, for an edge node E_i in $\{E_1, \ldots, R_m\}$, find the set of all critical paths

$$Critical(E_i) = \{\mathbf{A_1}(E_i), \ldots, \mathbf{A_k}(E_i)\},$$

where $\mathbf{A_l}(E_i)$ is the sequence of algorithms on the l-th critical path of the edge node E_i. From the first to the last non-trivial algorithm A_j in $\mathbf{A_l}(E_i)$, for $l = 1, \ldots, k$, apply the elementary step $\text{Elementary}_{E_i}(A_j)$ if possible and duplicate the algorithm A_j on a new node so that the value of

$$LevelofImproment(E_i, A_j, NewNode(A_j))$$

is maximal. Then update the set of all critical paths for all edge nodes and restart the process until the edge node E_i reaches the stop step. Then move to the next edge E_{i+1} and perform the same process until all edge nodes reach their stop steps. The pseudocode of the whole procedure is presented in Procedure 1.

This process is finite because applying an elementary process reduces the number of critical paths or the overall time to execute the critical path. Since the number of critical paths is finite (in the interval $[1, m]$), the overall time to execute the critical path

Procedure 1. Optimal Algorithm Allocation With Duplication Using Elementary Steps.

Input: Graph of algorithms, Architecture, and M is the optimal algorithm allocation minimizing time, [30]
Output: Optimal algorithm allocation with duplication.

```
for E_i ∈ E do
    CriticalPaths is the set of all critical paths of the edge node E_i.
    do
        Any_Duplicated = FALSE ▷ A variable used to identify whether an algorithm is duplicated or not.
        for CriticalPath_t ∈ CriticalPaths do
            for A_j ∈ CriticalPath_I do
                 𝔸_j = Class(A_j, CriticalPath_t) ▷ Algorithms in CriticalPath_I assigned to the same node as A_j is assigned to.
                 W = [].
                 for X ∈ V_n do
                     W = W ⊕ LevelofImproment(E_i, 𝔸_j, X) ▷ Concatenates LevelofImproment(E_i, 𝔸_j, X) one by one preserving its location.
                 k = arg max(W)
                 if W_k > 0 then
                     Apply Elementary_{E_i}(𝔸_j) ▷ Duplicate all the algorithms in 𝔸_j to the node k ∈ V_n.
                     M = M ∪ {(𝔸_j, k)}
                     Any_Duplicated = TRUE ▷ The value changes to TRUE because an algorithm is duplicated.
        Update CriticalPaths of the edge node E_i.
    while Any_Duplicated == TRUE
return M ▷ Optimal algorithm allocation with duplication minimizing time.
```

is finite[4], the preceding procedure can only be applied finitely many times.

Maximizing the term $LevelofImproment$ reduces the number of duplications necessary for the algorithm A_j to improve the value of $t_n^{res}(E_i)$.

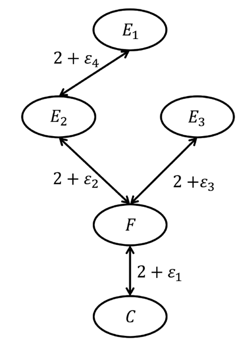
We show how the procedure works with a simple example. Given the graph of all algorithms, the architecture of the robotic network cloud system with communication instability, and the average execution time of each algorithm on each processing node, all in Fig. 5. In this figure, the task can be requested by E_1, E_2, and E_3 and all the algorithms, except virtual algorithms **0** and **1**, can be requested by any nodes.

Now we describe how the proposed procedure works. First, note that the optimal solution for the overall time of the system can be obtained by assigning all the tasks to the fog, where the overall times required by the edge nodes to obtain all the outputs

[4]The overall time of the critical path is in the interval $(0, InitialTime)$, where

$$InitialTime = t_n^{res}(E_i)$$

is the initial value of $t_n^{res}(E_i)$ before applying the first elementary step.



	Data	A_1	A_2	A_3	A_4	A_5	A_6	A_7
E_1	0	4	4	6	2	4	4	2
E_2	0	4	4	6	2	4	4	2
E_3	0	4	4	6	2	4	4	2
F	0	2	2	3	1	2	2	1
C	0	1	1	1.5	0.5	1	1	0.5

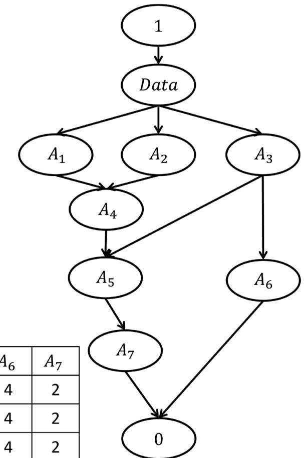


Fig. 5. Graph of all algorithms, architecture of the robotic network cloud system with communication instabilities, ε_i for $i = 1, 2, 3, 4$ are random variables following the folded normal distribution with mean 0 and variance 1, and the average execution time of each algorithm on each node. E_i's are edge nodes for $i = 1, 2, 3$, F is the fog, and C is the cloud.

TABLE I
THE AVERAGE RESPONSE TIME IN THE CLOUD SYSTEM'S NODES, AFTER APPLYING THE PROPOSED DUPLICATION PROCEDURE

	Time	Data	A_1	A_2	A_3	A_4	A_5	A_6	A_7
E_1	12.00	E_1	E_1	E_1	E_1	E_1	E_1	E_1	E_1
E_2	11.02	Fog	Fog	Fog	Fog	Fog	Fog	Fog	Fog
E_3	11.52	Fog	Fog	Fog	Fog	Fog	Fog	Fog	Fog

of all the algorithms are **15.26**, **11.02**, and **11.52** seconds for E_1, E_2, and E_3, respectively. This values are obtained by solving static algorithm allocation without duplication, [30]. Applying the proposed procedure implies that E_2 and E_3 are in the stop steps (because duplication cannot improve the minimum time). Since all algorithms are assigned to the same node, they should be duplicated to the same node. If we assign a copy all algorithms to E_1, E_2, E_3 and the cloud, we get a value of $\mathbf{3.26}$, $\mathbf{-3.05}$, $\mathbf{-10.73}$, and $\mathbf{-0.99}$ for $LevelofImproment$ respectively. The largest improvement occurs in the case where all the algorithms are duplicated on the edge node E_1, and then the edge node E_1 is in the stop step.

The results for duplication are shown in Table I. It shows that all algorithms should be allocated to the edge node E_1 and the fog node F to achieve the lowest task completion time, regardless of which node is to perform the task.

Now, according to the procedure proposed by [34], the following duplication can improve performance.

- duplicating $Data$ to the nodes that A_1, A_2, and A_3 are allocated to;
- duplicating A_3 to the nodes that A_5 and A_6 are allocated to.

Since all algorithms are assigned to the fog node, no duplication is performed. The results of algorithm duplication using the [34] procedure are shown in Table II.

TABLE II
THE AVERAGE RESPONSE TIME IN THE NODES OF THE CLOUD SYSTEM AFTER APPLYING THE [34] DUPLICATION PROCEDURE

	Time	Data	A_1	A_2	A_3	A_4	A_5	A_6	A_7
E_1	15.26	Fog	Fog	Fog	Fog	Fog	Fog	Fog	Fog
E_2	11.02	Fog	Fog	Fog	Fog	Fog	Fog	Fog	Fog
E_3	11.52	Fog	Fog	Fog	Fog	Fog	Fog	Fog	Fog

TABLE III
THE AVERAGE EXECUTION TIME OF EACH ALGORITHM ON EACH PROCESSING NODE

	Data	A_1	A_2	A_3	A_4	A_5	A_6	A_7
E_1	0	1	40	1	40	40	40	40
E_2	0	40	40	40	40	40	40	40
E_3	0	9	40	9	40	40	40	40
F	0	30	30	30	1	1	1	30
C	0	20	1	20	20	20	20	1

The results obtained in Tables I and II show that duplication procedures reduce the average response time of edge nodes. The edge node E_1 can respond at least four seconds faster when duplication is applied than in the case without considering duplication or using the procedure proposed by [34].

Currently, we know the solution of optimal allocation without duplication from [30]. Now, if the robot E_i starts to perform the task (the algorithms in the graph of all algorithms should be executed), then a critical path for the node E_i can be evaluated. Now we consider the algorithms on the critical path that belong to the same class (algorithms assigned to the same node), and duplicate them to other nodes to find out whether it reduces the overall time of the critical path or not. If there is some reduction, the duplication is performed. In this example, the optimal algorithm allocation without duplication is to allocate all algorithms to the fog F. Let us now consider the critical paths $A_1A_4A_5A_7$ and $A_2A_4A_5A_7$. Note that by construction, the response time is equal to the time to complete the critical paths, [30]. Also note that all algorithms are currently assigned to the fog node F. For E_2 and E_3, duplicating algorithms does not reduce the overall time. But for E_1, when all algorithms are duplicated on E_1, the time to complete the critical paths is reduced from 15.26 to 12 seconds. The critical paths remain critical, but their overall time decreases.

In this example, we compared the average response time of all the algorithms to all the edge nodes for the three cases where: (1) algorithm allocation is without duplication, (2) the procedure in [34], and (3) using our procedure. The final result shows that applying the duplication procedure using [34] does not change the performance and the result is the same as considering the algorithm allocation without duplication.

The following example is intended to show how the procedure works with a more complex example. The graph of all algorithms, and the architecture of the robotic network cloud system are the same as in Fig. 5. The average execution time of each algorithm on each processing node is shown in Table III.

Note that the optimal solution for the overall time of the system is to assign algorithms A_1 and A_3 to the edge node E_1, algorithms A_2 and A_7 to the cloud node C, and all the other algorithms to the fog node F. Using the static algorithm allocation without duplication, [30], the overall times required for the edge nodes to obtain all the outputs of all the algorithms are obtained as $\mathbf{36.94}$, $\mathbf{32.18}$, and $\mathbf{33.80}$ seconds for E_1, E_2, and E_3, respectively.

TABLE IV
THE AVERAGE RESPONSE TIME IN THE CLOUD SYSTEM'S NODES, AFTER APPLYING THE PROPOSED DUPLICATION PROCEDURE

	Time	Data	A_1	A_2	A_3	A_4	A_5	A_6	A_7
E_1	36..94	Fog	E_1	Cloud	E_1	Fog	Fog	Fog	Cloud
E_2	32.18	Fog	E_1	Cloud	E_1	Fog	Fog	Fog	Cloud
E_3	32.66	Fog	E_3	Cloud	E_3	Fog	Fog	Fog	Cloud

The application of the proposed procedure implies that E_1 and E_2 are in the stop steps (because duplication cannot improve the minimum time). But for E_3, if we assign E_2, E_3, the fog and the cloud a copy of the algorithms A_1 and A_3, the values for $Level of Improment$ will be respectively $\mathbf{-31.22}$, $\mathbf{0.14}$, $\mathbf{-13.40}$, and$\mathbf{-11.71}$. The greatest improvement occurs when algorithms A_1 and A_3 are duplicated on the edge node E_3 and then the edge node E_3 is in the stop step.

The results for duplication are shown in Table IV. It shows that A_1 and A_3 should be allocated to the edge nodes E_1 and E_3, A_2 and A_7 should be allocated to the cloud node, and all the other algorithms should be allocated to the fog node F to achieve the lowest task completion time, regardless of which node is to perform the task.

B. Mstep Procedure

We can modify the previous procedure to use smaller steps. Instead of reducing critical path time, this modification is done by duplicating a single procedure reducing the overall time for each edge node. Note that if instead of

$$t_n^{res} = \sqrt{\sum_{i=1}^{m} (t_n^{res}(E_i))^2}$$

in the optimization problem (1), we minimize $t_n^{res}(E_1)$ with the same constraints in the optimization problem (1), we find the optimal algorithm allocation solution that minimizes the overall time of the critical path of the edge node E_1. In this case, an elementary step applied to any algorithm will not reduce the overall time of any critical path of the edge node E_1, because the existence of an elementary step contradicts the fact that the algorithm allocation is minimal. Because otherwise, instead of the originally assigned node, we could choose the new node to which an algorithm duplication is assigned with the elementary step. This means that the edge node E_1 is in the stop step. For the edge nodes $E_i, i = 2, \ldots, m$, instead of applying the elementary steps, we could now find the optimal algorithm allocation for each and every one of them independently. In this way, we have the optimal algorithm allocation independently for all the edge nodes. The optimal algorithm allocation is the union of the solutions of all the edge nodes.

In [30] it is shown that the solution of the static allocation without duplication can be obtained in polynomial time. Since

Procedure 2. Optimal Algorithm Allocation Mstep Procedure

Graph of algorithms, G and Architecture $\text{Archi.}(E, F, C)$
$M = \emptyset$
for $E_i \in E$ **do**
  $M = M \cup \text{Solve}(G; \text{Archi.}(E, F, C) \mid \text{Objective} = \min\, t_n^{res}(E_1))$ ▷ Optimal algorithm allocation minimizing time, [30], by substituting the main objective with $t_n^{res}(E_1)$.

return M ▷ Optimal algorithm allocation with duplication minimizing time.

the Mstep procedure performs the static allocation without duplication for each edge node,[5] it can also be found in polynomial time.

The pseudocode of the whole procedure is presented in Procedure 2.

V. Experiments

The experiments are performed on a HP Laptop 15-dw2xxx with Intel Core i5 10th generation with processor Intel(R) Core(TM) i5-1035G1 CPU @ 1.19 GHz, RAM 16.0 GB, 64-bit operating system, and we used RStudio Version 1.4.1103 Â© 2009-2021, PBC and the R version 4.0.3 (2020-10-10) copyright Â© 2020.

For n robots, $n = 1, \ldots, 20$, the architecture has $n + 2$ nodes, an edge (communication) from the cloud to the fog node, and from the fog to at least one of the n robot nodes. To generate a random graph, we used Erdos-Renyi random graph generators, [36]. Since the architecture must correspond to a connected graph, we need at least $n - 1$ randomly placed edges between nodes. After each placement, we need to check whether the generated graph is connected or not (the number of edges of the architecture is randomly chosen from the set $\left\{n-1, \ldots, \frac{n(n-1)}{2}\right\}$), see Fig. 7. Once the generated graph is connected, we generate random delays from the folded normal distribution on the edges with parameters $(\mu = 0, \sigma = 1)$. For the graph of algorithms, we generate a random directed acyclic graph by randomly choosing the number of nodes N from $\{5, \ldots, 20\}$ with the constraint that the expected number of edges connected to the nodes is equal to $\frac{N}{3}$, see Fig. 6. The average execution time of each algorithm by all nodes are randomly chosen from the interval [0,5].

For the generated graph of algorithms and architecture, we solve the optimization problem (1) which gives us the optimal algorithm allocation without duplication (WoD), then apply the result of [34] which gives us the algorithm allocation with duplication, and finally apply our proposed Mstep procedure. The solutions of these procedures provide the values of the average overall times required to transmit all outputs of all algorithms to all robots, and then we compute the distance to the origin of all of these values. For the randomly generated architecture, due to the communication delays, we apply all procedures 10 times to solve the algorithm allocation, and we take the average of the results obtained by each procedure as the corresponding results of this architecture.

Recall that two graphs are isomorphic if and only if there is a bijection between vertices that preserves the connectivity of the edges. For more details on graph isomorphism, see [37]. To

[5]The overall execution time of the Mstep procedure is equal to the average execution time of the static allocation without duplication multiplied by the number of edge nodes.

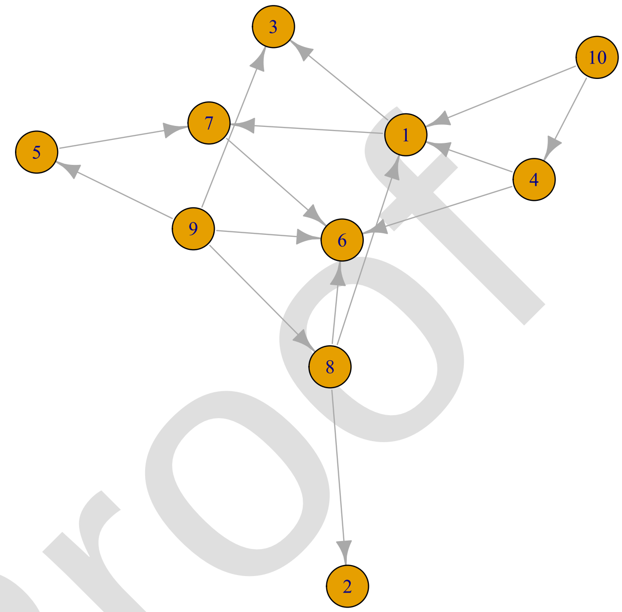


Fig. 6. Example of a randomly generated task using 10 algorithms.

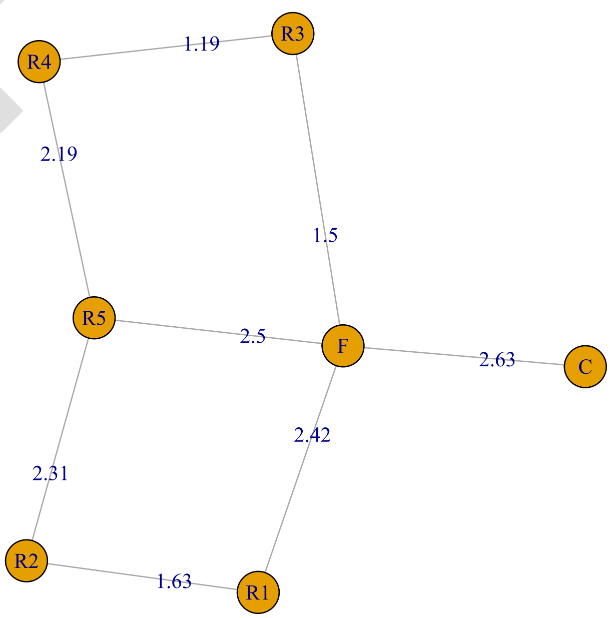


Fig. 7. Example of a randomly generated architecture with 5 robots. The values on the edge represent the average communication time between different nodes.

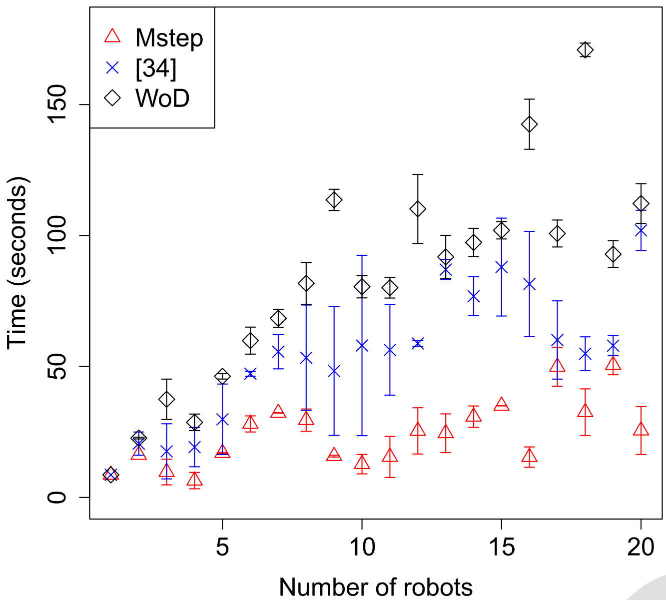


Fig. 8. Comparing the average overall times to transmit all outputs of all algorithms to all robots for a randomly generated graph of algorithms and randomly generated architectures with $n = 1, \ldots, 20$ robots. The bars are 99% confidence interval.

TABLE V
THE AVERAGE OVERALL TIMES FOR TRANSMITTING ALL OUTPUTS OF ALL ALGORITHMS TO ALL ROBOTS FOR A RANDOMLY GENERATED GRAPH OF ALGORITHMS AND RANDOMLY GENERATED ARCHITECTURES WITH $n = 1, \ldots, 20$ ROBOTS. sd IS THE STANDARD DEVIATION OF VALUES OBTAINED FOR RANDOMLY GENERATED GRAPHS

Number of robots	Mean time ($Mstep$)	Mean time ([34])	Mean time (WoD)	sd ($Mstep$)	sd ([34])	sd (WoD)
1	**8.64**	**8.64**	**8.64**	0	0	0
2	**16.27**	20.56	22.67	0.00	1.72	0.12
3	**9.70**	17.61	37.48	1.90	4.08	2.99
4	**6.43**	19.22	28.68	1.21	2.91	1.23
5	**16.96**	29.83	46.25	0.09	5.23	0.40
6	**28.07**	47.26	59.89	1.19	0.37	2.00
7	**32.29**	55.64	68.39	0.03	2.54	1.32
8	**29.53**	53.35	81.75	1.65	7.81	3.10
9	**15.79**	48.29	113.61	0.13	9.55	1.58
10	**12.71**	58.03	80.46	1.44	13.37	1.66
11	**15.47**	56.33	80.09	3.04	6.70	1.53
12	**25.42**	58.80	110.17	3.43	0.43	5.12
13	**24.51**	87.00	91.83	2.87	1.47	3.20
14	**30.85**	76.83	97.37	1.58	2.88	2.10
15	**35.01**	87.96	102.00	0.02	7.25	1.28
16	**15.41**	81.51	142.50	1.50	7.80	3.71
17	**49.88**	60.12	100.78	2.88	5.80	2.00
18	**32.55**	54.90	170.88	3.46	2.50	1.00
19	**50.55**	57.97	92.89	1.44	1.50	1.98
20	**25.55**	101.95	112.22	3.55	2.99	2.93

show that our procedure improves performance independently of the architecture, we randomly choose n non-isomorphic architectures.

Note that for the case $n = 1$, there is only one possible valid architecture that we need to consider. For $n = 2$, there are 4 valid architectures, two of which are isomorphisms. We have tested graph isomorphisms to avoid repeating the graph. For simplicity, in the case where the architecture has n robots, we considered and generated n random non-isomorphic graphs. The results are shown in Fig. 8. It shows that our proposed procedure (Mstep) outperforms both [34] and WoD in minimizing the average completion time of all algorithms when the task is performed by any of the robots. It shows that our Mstep procedure reduces the response time. The values of the results are given in Table V.

Note that once the system finds the solution to duplicate algorithms, it will use that solution to complete tasks as long as the problem does not change. Therefore, even a small improvement in task completion times will add up, which means that the number of completed tasks in a system using our duplication using Mstep procedure will be larger than the same system using the other procedures.

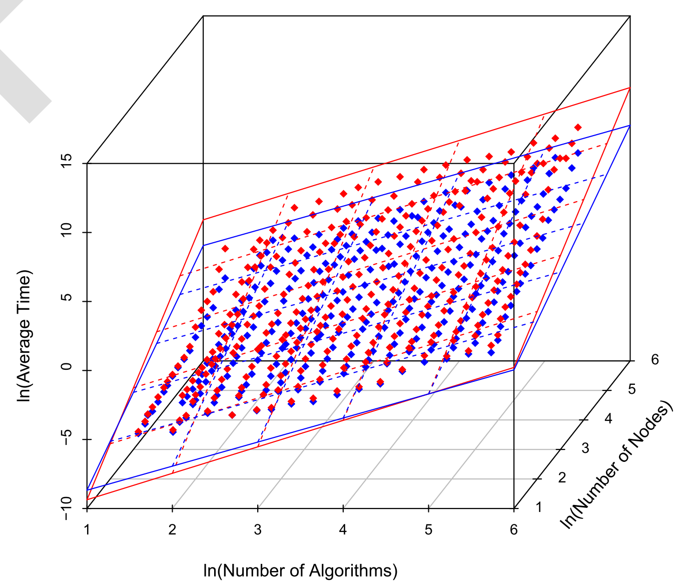


Fig. 9. The average execution time (in seconds) of procedures [34], in blue, and ours as functions of number of processing units and number of algorithms, in red. The planes are the fits obtained by the linear regressions with the $R^2 = 0.9788$ for [34], the $R^2 = 0.9725$ for ours.

VI. SCALABILITY ANALYSIS

For a given architecture, like other procedures searching for the longest path in a graph, our procedure's time complexity is in NP.

We conducted an experiment where we randomly produced the graph of all algorithms and randomly built the architecture for a particular number of nodes to assess the scalability of our procedure and compare it with [34] and static allocation without duplication. For each of the ten randomly generated architectures, the number of algorithms is determined, and ten algorithm graphs are generated at random for each architecture. In Fig. 9, we show the average amount of time it takes to solve 10 graphs of algorithms. The graph illustrates a linear relationship between the average time and the number of algorithms to allocate as well as a linear relationship between the average time and the number of nodes. All axes are in logarithmic scale. Hence the time complexity of our procedure is polynomial.

VII. Conclusion

Duplication of algorithms can improve the performance of robotic network cloud systems. We proposed two procedures for static algorithm allocation for robotic network cloud systems that determine which algorithms should be duplicated and where they should be allocated to. The advantages of our procedures over the procedure in [34] are that they only consider communication times for duplication, not nodes with different execution times, and that their procedure includes necessary (but not sufficient) conditions for task duplicability. However, since the conditions in [34] are not sufficient, there may be some algorithms whose duplication can improve the performance of the system which are not used in that work, but our procedures work for any architecture and provide the nodes to which duplicated algorithms should be assigned to.

Static allocation with duplication (Mstep) must be performed *only* once, while for all other procedures, static allocation without duplication must be performed *at least* once. In our Mstep procedure, static allocation without duplication needs to be solved multiple times (depending on the number of robots), which may cause delays in the start time of the actual task performance by the robots. However, since our Mstep procedure reduces the response time, as shown by the experimental results in Fig. 8, the system completes more tasks in the long run than using other procedures.

We conducted experiments with random architectures and algorithms and compared our results with those proposed in [34] and confirmed the improvements in our proposal.

References

[1] E. Prassler and K. Kosuge, *Domestic Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1253–1281.

[2] Y. Xu, H. Qian, and X. Wu, *Household Service Robotics*. Amsterdam, The Netherlands: Elsevier, 2014.

[3] S. Chatterjee, R. Chaudhuri, and D. Vrontis, “Usage intention of social robots for domestic purpose: From security, privacy, and legal perspectives,” *Inf. Syst. Front.*, Sep. 2021. [Online]. Available: https://doi.org/10.1007/s10796--021-10197-7

[4] L. T. Ross, S. W. Fardo, and M. F. Walach, *Industrial Robotics Fundamentals: Theory and Applications*, 3rd ed., Homewood, IL, USA: Goodheart-Willcox., 2017.

[5] I. International Federation of Robotics IFR worldwide. (2020) International Federation of Robotics IFR, industrial robots, International federation of robotics IFR, industrial robots, 2020. [Online]. Available: https://www.ifr.org/industrial-robots

[6] R. Pillai, B. Sivathanu, M. Mariani, N. P. Rana, B. Yang, and Y. K. Dwivedi, “Adoption of AI-empowered industrial robots in auto component manufacturing companies,” *Prod. Plan. Control*, vol. 33, no. 1, pp. 1–17, 2021. [Online]. Available: https://doi.org/10.1080/09537287.2021.1882689

[7] P. Springer, *Military Robots and Drones: A Reference Handbook*, Santa Barbara, CA, USA: ABC-CLIO, 2013.

[8] V. Nath and S. E. Levinson, *Autonomous Military Robotics*. Berlin, Germany: Springer Publishing Company, 2014.

[9] O. Rosendorf, “Predictors of support for a ban on killer robots: Preventive arms control as an anticipatory response to military innovation,” *Contemporary Secur. Policy*, vol. 42, no. 1, pp. 30–52, 2021. [Online]. Available: https://doi.org/10.1080/13523260.2020.1845935

[10] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed., Berlin, Germany: Springer Publishing Company, 2016.

[11] X. V. Wang and L. Wang, “A literature survey of the robotic technologies during the COVID-19 pandemic,” *J. Manuf. Syst.*, vol. 60, pp. 823–836, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0278612521000339

[12] G. Hu, W. P. Tay, and Y. Wen, “Cloud robotics: Architecture, challenges and applications,” *IEEE Netw.*, vol. 26, no. 3, pp. 21–28, May/Jun. 2012.

[13] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, “A survey of research on cloud robotics and automation,” *IEEE Trans. Automat. Sci. Eng.*, vol. 12, no. 2, pp. 398–409, Apr. 2015.

[14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the Internet of Things,” in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, New York, NY, USA, 2012, pp. 13–16. [Online]. Available: https://doi.org/10.1145/2342509.2342513

[16] S. Alirezazadeh and L. A. Alexandre, “Dynamic task allocation for robotic network cloud systems,” in *Proc. 19th Int. Conf. Ubiquitous Comput. Commun.*, 2020, pp. 1221–1228.

[17] S. Alirezazadeh and L. A. Alexandre, “Improving makespan in dynamic task scheduling for cloud robotic systems with time window constraints,” *Cluster Comput.*, Sep. 2022. [Online]. Available: https://doi.org/10.1007/s10586--022-03724-x

[18] R. Burkard, M. Dell’Amico, and S. Martello, *Assignment Problems*, Philadelphia, PA, USA: SIAM, 2012. [Online]. Available: https://epubs.siam.org/doi/pdf/10.1137/1.9781611972238

[19] M. C. Gombolay, R. J. Wilcox, and J. A. Shah, “Fast scheduling of robot teams performing tasks with temporospatial constraints,” *IEEE Trans. Robot.*, vol. 34, no. 1, pp. 220–239, Feb. 2018.

[20] L. E. Parker, “ALLIANCE: An architecture for fault tolerant multirobot cooperation,” *IEEE Trans. Robot. Automat.*, vol. 14, no. 2, pp. 220–240, Apr. 1998.

[21] W. Chen, Y. Yaguchi, K. Naruse, Y. Watanobe, and K. Nakamura, “QoS-aware robotic streaming workflow allocation in cloud robotics systems,” *IEEE Trans. Serv. Comput.*, vol. 14, no. 2, pp. 544–558, Mar./Apr. 2021.

[22] J. He, M. Badreldin, A. Hussein, and A. Khamis, “A comparative study between optimization and market-based approaches to multi-robot task allocation,” *Adv. Artif. Intell.*, vol. 2013, 2013, Art. no. 256524. [Online]. Available: https://doi.org/10.1155/2013/256524

[23] L. Wang, M. Liu, and M. Q. Meng, “A hierarchical auction-based mechanism for real-time resource allocation in cloud robotic systems,” *IEEE Trans. Cybern.*, vol. 47, no. 2, pp. 473–484, Feb. 2017.

[24] L. Wang, M. Liu, and M. Q. Meng, “Hierarchical auction-based mechanism for real-time resource retrieval in cloud mobile robotic system,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2014, pp. 2164–2169.

[25] M. Gombolay, R. Wilcox, and J. Shah, “Fast scheduling of multi-robot teams with temporospatial constraints,” 2013. Q4

[26] L. Wang, M. Liu, and M. Q. Meng, “Real-time multisensor data retrieval for cloud robotic systems,” *IEEE Trans. Automat. Sci. Eng.*, vol. 12, no. 2, pp. 507–518, Apr. 2015.

[27] N. Tsiogkas and D. M. Lane, “An evolutionary algorithm for online, resource-constrained, multivehicle sensing mission planning,” *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 1199–1206, Apr. 2018.

[28] M. U. Arif, “An evolutionary algorithm based framework for task allocation in multi-robot teams,” in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 5032–5033.

[29] Z. Cheng, P. Li, J. Wang, and S. Guo, “Just-in-time code offloading for wearable computing,” *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 1, pp. 74–83, Mar. 2015.

[30] S. Alirezazadeh, A. Correia, and L. A. Alexandre, “Optimal algorithm allocation for robotic network cloud systems,” *Robot. Auton. Syst.*, vol. 154, 2022, Art. no. 104144. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889022000835

[31] S. Li, Z. Zheng, W. Chen, Z. Zheng, and J. Wang, “Latency-aware task assignment and scheduling in collaborative cloud robotic systems,” in *Proc. IEEE 11th Int. Conf. Cloud Comput.*, 2018, pp. 65–72.

[32] S. Alirezazadeh and L. A. Alexandre, “Optimal algorithm allocation for single robot cloud systems,” *IEEE Trans. Cloud Comput.* vol. 11, no. 1, pp. 324–335, First Quarte 2023.

[33] S. Mostafavi and V. Hakami, “A stochastic approximation approach for foresighted task scheduling in cloud computing,” *Wireless Pers. Commun.*, vol. 114, no. 1, pp. 901–925, Sep. 2020. [Online]. Available: https://doi.org/10.1007/s11277--020-07398-9

[34] M. Orr and O. Sinnen, “Integrating task duplication in optimal task scheduling with communication delays,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 10, pp. 2277–2288, Oct. 2020.

[35] M. Mistry, D. Letsios, G. Krennrich, R. M. Lee, and R. Misener, “Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded,” 2018.

[36] P. Erdős and A. Rényi, “On the evolution of random graphs,” *Pub. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.
[37] J. L. Gross, J. Yellen, and M. Anderson, *Graph Theory and its Applications*, 3rd ed., Boca Raton, FL, USA: CRC Press, 2019.

Saeid Alirezazadeh received the PhD degree from Centro de Matemática - Universidade do Porto, Porto, Portugal, in 2015. He was a postdoctoral researcher with Instituto Superior de Agronomia, Lisbon, Portugal, from 2016 to 2019. He was a post-doctoral researcher with C4-Cloud Computing Competence Center, Universidade da Beira Interior, Covilhã, Portugal, from 2019 to 2022. His research interests include algebraic graph theory, semigroups, automata, and tree languages, optimal scheduling of cloud robotics systems, and theoretical modeling of dynamical systems and stochastic processes. He is currently a postdoctoral researcher with Physikalische und Theoretische Chemie, Universität, Graz, Austria.

Luís A. Alexandre received the BSc, MSc and PhD degrees from the University of Porto, in 1994, 1997, and 2002, respectively, and an habilitation on computer science and engineering from UBI, in 2013. His research interests are neural networks, computer vision and their applications, particularly to robotics. He has been PI of several research projects with total budgets over 2M€, involving both academia and industry, has co-chaired 2 international conferences, participated in more than 50 international conference program committees and has been an evaluator for the A3ES (portuguese agency for the Quality Assurance in Higher Education). He was also a member of the governing board of both the International Asssociation for Pattern Recognition and of the European Neural Network Society and the president of the Portuguese Association for Pattern Recognition.