Evaluation of Objective Function Descriptions And Optimization Methodologies For Task Allocation In A Dynamic Fog Environment

Reinout Eyckerman, Siegfried Mercelis, Johann Marquez-Barja and Peter Hellinckx
IDLab - Faculty of Applied Engineering
University of Antwerp - imec
Sint-Pietersvliet 7, 2000 Antwerp, Belgium
Email: reinout.eyckerman@uantwerpen.be

***Abstract*—Industry, healthcare, and various other sectors are rapidly adopting the Internet of Things to drive information and automation systems. However, as the number of devices increases, the number of information sent over the network increases as well, inducing network congestion and a potential latency increase. To ensure that demanding applications, such as smart vehicles, are supported in the current network infrastructure, we provide a general methodology of distributing software from the cloud toward the edge, reducing multiple objectives such as latency. In this research we define several problems in multi-objective distribution scenarios, and compare several methodologies for defining and solving the problem. Additionally, we propose a method for decreasing the problem complexity, improving performance with only slightly reduced accuracy.**

ACKNOWLEDGEMENT

This research received funding from the Flemish Government (AI Research Program).

I. INTRODUCTION

The Internet of Things (IoT) has become an intrusive technology, taking both consumers as businesses by storm. An example is the Industrial Internet of Things, which is based on bringing IoT intelligence to the industry, and is a core pillar of the Industry 4.0. Automated Guided Vehicles (AGVs) are one such use case, which combine local sensor information together with a global environment model, provided by cloud services using cameras and other systems. Other examples are storage facilities automating stock handling using RFID.Most major cloud providers support services that work jointly with software stacks built upon IoT devices, allowing combining the IoT sensing and actuation with the cloud's massive compute capabilities. This allows compact and resource-constrained IoT devices with a massive computing backend. However, the IoT - Cloud paradigm has drawbacks, such as the detrimental effect on the network. As IoT devices are continuously connected, they start to congest the network and increase the latency. According to Cisco, 850 ZB of data is predicted to be generated in 2021, more than triple the 220 ZB generated in 2016 [1]. An up-scaling of data centers and network links is required to process this increasing quantity. A latency increase is unacceptable in low-latency environments, such as the previously defined AGV scenario, where it can cause extra costs due to devices crashing into each other. Such issues cause a steady research into improving the current network infrastructure, such as 5G networking, which encompasses fields such as Software Defined Networks and optimizing network traffic routes. Fog computing is another solution which intends to move software currently being ran in the cloud towards closer computing resources, using personal computers and routing/switching devices such as the Cisco IOx devices [2]. Distributing tasks over the fog allows reducing the load on the network by using pre- and post-processing tasks, ensuring only the essential data is transferred, as shown in Fig. 1. Moreover, distribution aids in efficiently using all the available device resources, as these often have excess compute available.

However, research in this area has focused on solving specific use-cases. The lack of general methodology makes it difficult to find a generally optimal solution across various network types. To this goal, we propose a general coordinator service which considers the device contexts and required objectives. These objective models can be changed with minimal effort, allowing the rapid deployment on different kind of networks with different requirements and objectives. It will continuously optimize the task placement over the network, so that optimal resource usage with an optimal performance is achieved. Moreover, we differentiate between Multi-Objective Optimization (MOO) and Single-Objective Optimization (SOO), and propose an approach to make large problems more scaleable. The remainder of the paper is structured as follows: In Section II we describe the current State of the Art related to this research. We then continue into Section III where the problem is formulated. In Section IV we state our methodology, and in Section V we showcase our achieved results. This is followed by a Conclusion in Section VI. Finally, Section VII describes future research directions related to this research.

II. STATE OF THE ART

Fog and Edge computing were developed as a response to the need of keeping IoT applications low-latency while maintaining the low compute capabilities of the IoT devices themselves to press costs. Manual software placement, a key

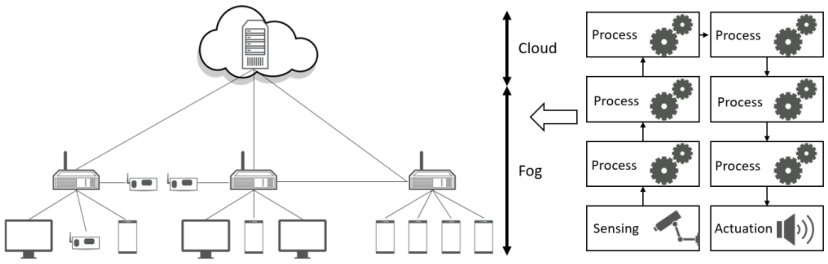

Fig. 1. Task Mapping Problem

issue, is time-consuming and error prone. Additionally, this problem has many variables and complexities, as defined in previous research [3]. The placement problem is known to be an NP-hard problem [4], where the complexity of finding a solution increases as the amount of components or machines in the network rises. The best solution has the optimal location of software across the network with reduced network load and improved resource usage. Sub-optimal placements could even increase the overhead compared to regular cloud computing.

Task allocation in fully distributed systems has additional issues, such as the lack of a global control unit and resource distribution complexities. If the network on which the components are placed is static in nature, the software placement happens once, resulting in a "set and forget" approach. However, few networks are actually static, and as the network changes, the optimal component placements deteriorate over time. Similarly, new applications can continuously arrive to the network to be distributed. However, the context in which these computations take place also changes continuously. Context-awareness is a broad field with plenty complexities. If a local device decides to back everything up to a cloud server, the available bandwidth decreases. If a user manually places a task on a device, the coordinator should reduce the task amount dynamically assigned to this device, to prevent the device overloading. We will tackle the previously defined complexities, such as complexity, dynamic behaviour and context-awareness, in the following subsections.

A. Frameworks

Fog applications have close ties to distributed systems, which are harder to develop and maintain than monolithic applications. Frameworks such as AIOLOS by De Coninck et al. attempt to lower the development cost of such distributed software [5]. This framework enables the deployment of component-based applications across heterogeneous devices, without having to manage the communication channels. A related framework is Distributed Uniform Streaming Framework (DUST), which adds a transparent communication to lower the development cost, handling the transport complexities for the executing program [6]. This transparency allows developers to pay more attention to their core logic. As this framework is configuration-based and allows for easy and modular communication stacks, we will use this framework as the application chain base. This high level of control enables optimal configuration of the stack by the coordinator.

B. Placement Techniques

Previous research has shown that centralised and distributed task allocation coordination techniques both have their own place in the world of distributed task allocation [7]. We described requirements for both distributed and centralised coordination techniques, addressing issues such as the finding of a global optimum, scalability to large-scale networks and a limited resource consumption so that it can run in resource-constrained networks. We will discuss both techniques below.

1) Distributed Algorithms: Centralised coordination is especially efficient in smaller scale static networks where efficiency is essential, whereas distributed coordinators are more dynamic, but have problems with finding good optima. Vanneste et al. [6] proposed a distributed task coordination protocol. They applied the Contract-Net protocol for decentralized task allocation, considering resource constraints such as battery level. However, dynamic approaches have a hard time finding optima. For this, previous research proposed a hybrid approach using Ant Colony Optimisation, where ants traverse the network and build up a task placement suitability map of the traversed machines, leaving behind pheromones for other ants to improve their placement [7]. The approach might still be too slow for highly dynamic networks, however, but is more scalable than a centralised coordinator. However, as the complexities, such as Multi-Objective Optimization (MOO), are considerably higher, we will first focus on centralised algorithms for optimal placement.

2) Centralised Algorithms: Previous research has provided a static centralised coordination service deploying software for fog networks [8]. Several approaches into task placement heuristics were considered on tree graph structures, considering the network shape. Similarly, Tang et al. [9] used machine learning algorithms to optimise task migration policies. These policies are used once all the tasks are deployed, so that there is no additional overhead of unnecessarily moving tasks. Using centralised coordination, a search algorithm can search towards a global optimum as the entire network state is known. One popular metaheuristic for this is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) Genetic Algorithm, proposed by Deb et al. [10]. This bio-inspired algorithm uses evolution through reproduction, mutation and

elitism for the desired results. Alternatives exist, such as Multi-Objective Particle Swarm Optimization (MOPSO), where a swarm of particles is simulated, which moves toward the objective space's optima. Most MOO algorithms tend to work in continuous search spaces however, whereas our search space is discrete. This is because a task only maps to a single device. Discrete versions of most algorithms exists, such as the continuous-discrete FC-MOPSO algorithm by Mokarram et al., which we implemented in this paper [11]. We implemented these centralised algorithms, defined potential issues and showcase them in the results.

C. Models

To determine the best placement, a model of the both the application and the network is required. The requirements of the application model should be mappable to the network capabalities. These are closely bound to the context of the application and the network.

1) Application Model: Application requirement examples are Worst-Case Execution Time (WCET) and the maximally allowed latency. Recent research has shown promising techniques for measuring WCET. Huybrechts et al. [12] propose a novel hybrid WCET measuring technique, taking static measurements based on source code and dynamic measurements by executing software. These are combined using Machine Learning to do predictions on code bases, enabling measurements on large applications. Krommydas et al. [13] define the OpenDwarfs benchmark for measurements of heterogeneous computing capabilities. It allows comparing multiple processor types (CPU, GPU, FPGA) based on several idioms such as execution time and data transfer. Such research can define the application metrics, essential for this research. These metrics are then considered so that the objectives and constraints can be determined. An issue is the large device heterogeneity and metric difference. This can be solved by measuring across the set of available devices, creating lookup tables.

2) Context: Distribution techniques only work efficiently if they consider the context. This tells us more about the specific device, software and other variables. Liu et al. [14] describe a context as a way of processing data quality. This data quality is a sensor output quality measure, and this context manages to transform it into information, which can then be used by the application. This is closely related to the actual requirements for the application, as a dirty camera with a part blacked out might still be functional for the application. Baldauf et al. provide us with an extensive survey of context in relation to applications [15]. They also make a differentiation between physical sensors (hardware), virtual sensors (software) and logical sensors (sensor fusion). These, however, are application contexts. The device context should be considered as well, which might change due to environmental factors (e.g. heat) or external processing requirements. Moreover, as networks are dynamic systems, the throughput also changes. We will define how these contexts come into play when determining an optimal placement. We attempt to optimize the coordination techniques for centralised coordination using more fundamental techniques, and compare several methods.

III. Problem Statement

The goal of task distribution coordination is to create a task to device mapping in such a way that certain objectives, such as energy usage and latency, are optimized while adhering to the device constraints, such as available memory or bandwidth, as shown in Fig. 1. This has proven to be an NP-hard problem [16]. Considering dynamic device locations and continuous link variations further increases this complexity. However, the devices and the software deployed on them can change dynamically as well. Depending on user requests, different software can be deployed and undeployed, and depending on the information the software receives as input, the resource requirements can also change drastically. Moreover, the context also plays a large role. We define two context types for this problem, of which we will tackle the first in this research.

1) Resource Context: Within the resource context, device capabalities can change. For example, users can put software on devices in multi-tenant scenarios. This reduces the tasks that can be deployed on the device. External factors such as battery level or heat can also reduce the amount of computations that the device can do, and hardware failures can further influence this. Moreover, the network itself is a dynamic entity, as data streams continuously change conform to the user demands.

2) Application Context: In the application context, data quality is an important factor: depending on environmental factors, the sensors can produce less useful input, such as a camera at night or a microphone when it is windy. This depends on the application's requirements: a camera still detects light at night, which might be the application's goal. Moreover, the application can choose to use different sensors depending on the environment, which has an effect on the application's resource requirements. With this, we show that a large set of variables influence this problem. We will next define and tackle several variables.

A. Network & Application Models

The network is considered dynamic, with device relocation, changing links and even context shifts. These context shifts signify a change in, for example, load on devices by applications run by users, causing the potential need to redistribute an application to ensure optimal resource usage. A software distribution coordinator has to consider these numerous factors to achieve an optimal placement. This is a complex task, of which we will attempt to solve multiple parts, such as optimal placement and dynamic networks. The network has been modeled as a weighted directed graph, defined as $\boldsymbol{N} = \{M, L\}$. Here, M represents the set of vertices/machines $m \in M$. L is a set of directed edges $l \in L$. The directed links allow the independent modeling of up- and download link metrics, which are unnecessarily the same. This holds especially for fog networks, where the Internet Service Provider often throttles the upload speed. The considered

device metrics and constraints are shown in Table I. Machine constraints prevent processor, memory and Network Interface Cards (NICs) overloading. Link constraints use bandwidth as a constraint, preventing link overuse inducing extra latency, interference or even failures. The changing device context also manipulates these constraints. A device placed under full load cannot guarantee that any additional tasks work efficiently. The application placed over the network is modeled as a

TABLE I
USED NOTATIONS

Notation	Parameter
m_{cpu}	Available CPU capabilities of device m
$m_{cpu,e}$	Energy requirements of the CPU of device m
m_{mem}	Available memory capabilities of device m
$m_{mem,e}$	Energy requirements of the memory of device m
m_{tx_i}	Available transmission capabilities of NIC i of device m
m_{rx_i}	Available receiving capabilities of NIC i of device m
$m_{trx,e}$	Energy requirements of NIC i of device m
m_{stor}	Available storage capabilities of device m
$m_{stor,e}$	Energy requirements of the storage of device m
m_e	Energy cost of device m
l_{bw}	Available bandwidth of link l
l_{lat}	Induced latency of link l
t_{cpu}	CPU requirements of task t
t_{mem}	Memory requirements of task t
$t_{wcet,m}$	WCET of task t on device m
t_{wcetd}	WCET deadline of task t
t_{stor}	Storage requirements of task t
t_{tx_i}	Transmission requirements on communication link i of task t
t_{rx_i}	Receiving requirements on communication link i of task t
c_{bw}	Bandwidth requirements of communication link c
c_{lat}	Maximally allowed latency of communication link c

weighted directed graph as well, defined as $\boldsymbol{A} = \{T, C\}$, where T is the set of vertices/tasks $t \in T$ which are to be executed, and C the set of edges/communication links $c \in C$ between each task. Multiple edges between each service represent multiple "communication channels". This allows the modeling of different communication parts, such as using different channels for data or control. The requirements of task t are the worst-case resource consumption parameters, such as WCET. These can be calculated using existing frameworks such as the COBRA tool [12]. Similarly, the requirements of communication link c are constraints such as the required bandwidth. Only stateless tasks are considered to remove the state information communication and retaining complexity. The required software is available on each device, removing the need to transfer it from an application source.

B. Objective Function

To optimize the placement, we will optimize the objective functions displayed in Table II. We minimise WCET across devices, energy cost per used resource across devices, and reduce the bandwidth used over the network. For latency, we minimise the induced latency between every task. Optimising multiple objectives is a Multi-Objective Optimization (MOO) problem. As these objective functions are often conflicting, there often is no single best solution. Instead, a pareto front is found, a front in the objective space where all solutions are pareto-optimal. This is defined as follows [17]:

Definition 1. An objective vector $f(\mathbf{x}^*)$ based upon decision vector $\mathbf{x}^* \in S$ is Pareto-Optimal if there does not exist another decision vector $\mathbf{x} \in S$ where $f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$ for all $i = 1, 2, \ldots, W$ and $f_j(\mathbf{x}) < f_j(\mathbf{x}^*)$ for at least one j.

Correspondingly, we define the nadir and utopia point [17]:

Definition 2. An objective vector $\mathbf{z^{nad}} = (z_1^{nad}, z_2^{nad}, \ldots, z_W^{nad})$ where the j-th element is taken from a pareto-optimal point where the worst value for f_j is achieved is called a Nadir Point.

Definition 3. An objective vector $\mathbf{z^o} = (z_1^o, z_2^o, \ldots, z_W^o)$ where the j-th element is taken from a pareto-optimal point where the best value for f_j is achieved is called a Utopia Point.

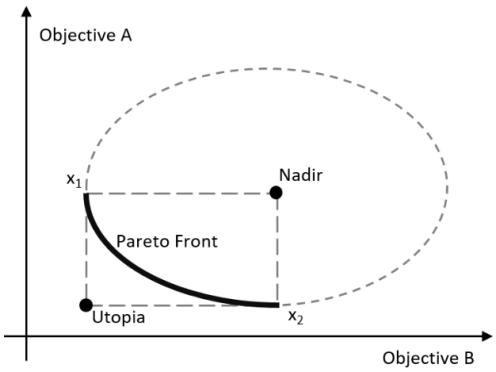

Fig. 2. Visualisation of a search space and its corresponding Pareto Front.

The utopia and nadir points are visualised on Fig. 2, where the dotted oval signifies the objective space. These definitions show that the nadir and utopia points depend on the pareto front. Do note that the utopia point is generally not attainable due to conflicting objective functions. Moreover, there are a set of constraints we have to adhere to, displayed in Table III. Any solution not adhering to these constraints is considered an infeasible solution. With regards to MOO, these constraints can be considered as hard objectives, being either met or unmet. This allows comparing solutions based on the amount of satisfied constraints, ensuring that the most viable solution is found, even if constraints cannot be met. Constraint handling has been implemented according to Deb et al. [10]. They propose adding a constraint metric to every solution, where non-constrained solutions automatically dominate constrained solutions, and constrained solutions dominate each other depending on how far the constraints are exceeded. This is done by summing how much percent the placement exceeds the device's available resources. We will now define multiple methods of handling objectives and constraints.

TABLE II
OBJECTIVE FUNCTIONS

Objective	Function
WCET	$\sum_{m}^{M}\sum_{t}^{m} t_{wcet,m}$
Energy	$\sum_{m}^{M}\sum_{t}^{m} m_{t,e} * m_{ec}$
Bandwidth	$\sum_{c}^{C}\sum_{l}^{c_{route}} c_{bw}$
Latency	$\sum_{c}^{C}\sum_{l}^{c_{route}} l_{lat}$

TABLE III
CONSTRAINTS

Constraint	Function
WCET	$\forall m \epsilon M, t \epsilon T : t_{wcet,m} \leq t_{wcetd}$
CPU usage	$\forall m \epsilon M : m_{cpu} \geq \sum_{t}^{m_T} t_{cpu}$
Memory usage	$\forall m \epsilon M : m_{mem} > \sum_{t}^{m_T} t_{mem}$
Storage usage	$\forall m \epsilon M : m_{stor} > \sum_{t}^{m_T} t_{stor}$
NIC outgoing usage	$\forall m \epsilon M, NIC \epsilon m : NIC_{tx} > \sum_{t}^{m_T} t_{tx,c}$
NIC incoming usage	$\forall m \epsilon M, NIC \epsilon m : NIC_{rx} > \sum_{t}^{m_T} t_{rx,c}$
Bandwidth	$\forall l \epsilon L : l_{bw} \geq \sum_{l}^{c_{route}} c_{bw}$
Latency	$\forall c \epsilon C : c_{maxlat} \geq \sum_{c}^{l_{route}} l_{lat}$

IV. METHODOLOGY

There are multiple ways to tackle MOO Problems. Each, however, has its own benefits and drawbacks. We will now define and compare two, and look into solution space reduction and solution re-use to reduce the problem complexity.

A. Solution Space Exploration

We will compare MOO with SOO within the context of task allocation. There is a certain added complexity in finding proper solutions for MOO problems. As there are multiple pareto-optimal points, a Decision Maker (DM) has to determine which points are most relevant to him. With this in mind, we have three ways of solving MOO problems:

- A priori: User preferences are defined in advance, and the algorithm works with these.
- A posteriori: The algorithm provides the pareto front, or a part of it, and the DM selects the most relevant solution.
- Interactive: The algorithm and the DM interact to find a fitting solution.

In this case, a priori methods are the most interesting, as it allows the system to work autonomously, whereas the other tend to require interference of the DM. Two methods will now be defined for finding a solution.

1) Weighted Sum: The first solution to tackling the MOO problem uses the Weighted Sum method, where each objective gets multiplied with its user preference, and then summed together [18]. This results in a single objective function in which we then search for an optimum. However, such summation brings along some complexities. To properly execute the weighted sum method, each objective function that is optimized is to be normalised. This removes its dimensions, so that the resulting single objective function considers each objective equally, ensuring that no objective has more weight than the other purely due to its magnitude. Moreover, normalizing these objectives helps the DM with selecting appropriate weights, as each weight now can have the same order of magnitude. To normalize the objectives, the previously defined nadir and utopia point are required, allowing us to normalize them using the following equation [19]:

$$\bar{u_i} = \frac{u_i - z_i^o}{z_i^{nad} - z_i^o} \tag{1}$$

Here, $\bar{u_i}$ represents the normalized objective i. The goal for solving a MOO problem is to find either a single point on the pareto front which best matches user preferences, or finding the entire pareto front. However, this is a Catch-22 situation: to find a pareto optimum using a single objective function, the nadir and utopia points are required. But to find these points, the pareto front is required.To find the nadir and utopia points, required for single-objective optimization, we apply research done by Deb et al. [17]. They propose an adaptation of the NSGA-II algorithm. The original NSGA-II algorithm attempts to keep its population as diverse as possible by preventing population crowding, allowing for more exploration. The proposed adaptation changes this crowding function, by preferring the individuals closer to the pareto front extremes, as this is closer to the values required for the nadir point. These extremes are shown as x_1 and x_2 for the two objectives visualised in Fig. 2. The population gets spread over the pareto front borders, maintaining its diversity [20]. This provides us with the required population so that an estimate of the nadir and utopia points can be created. To normalize solutions, we use the following objective equation, where o_k corresponds with the objective functions defined in Table II, and w_k represents the weight for the objective as defined by the DM.

$$C = \sum_{k}^{\#Objectives} o_k * w_k \tag{2}$$

This requires the parameters to be normalized using Eq. 1. This is then optimised using a regular Genetic Algorithm (GA).

2) Evolutionary Multi-Objective Optimisation: Our second approach is the use of an Evolutionary Multi-Objective Algorithm (EMOA). For this we will use both NSGA-II [10], a widely-used baseline, and the FC-MOPSO swarm optimizer [11]. NSGA-II is a genetics inspired algorithm, using reproduction, mutation and elitarianism as functions to grow the population towards the desired goal. FC-MOPSO is particle swarm optimization algorithm, with the swarm searching through the space. These techniques are aimed at multi-objective optimization, but have a few drawbacks. One is the difficulty to embed user preferences into the search,

as each objective function gets updated separately. Wang et al. overcame this problem by adding a user preference embedding, essentially applying the weighted sum method inside NSGA-II [21]. However, it can reduce the effectiveness, as preferring solutions with good user weights can reduce the solution diversity.Without user preference embedding, these algorithms work a posteriori, where it finds a part of the pareto front. The DM must find a solution from this front which he assumes to be best conform to his preferences. This approach is not preferred, as the coordinator is to work completely autonomously. We however use this a posteriori approach to find a pareto front, and then use the weighted sum method to compare the solutions. The DM assigns a preference weight to each objective function, using for example the ranking method. Once the pareto front is determined, the objective function weights are normalized by creating a nadir and utopia point from the resulting solution. The weights are then multiplied with the normalized results, which are then summed together, resulting in a weighted sum as defined in Eq. 2. From the set of pareto-optimal placements the most preferential placement is found by minimizing this function.

B. Graph matching

As the model contains many variables, especially in advanced network models, we propose applying pre-processing to reduce the search space complexity. We simplify the problem by applying graph coarsening to the application graph, reducing number of tasks and decreasing the algorithm search area. This is implemented using a greedy maximum weight matching algorithm. The tasks are aggregated on the edges which have the largest bandwidth requirements, the lowest latency requirements and the lowest device requirements. This ensures that the high load links are merged, while the merged tasks can still run on almost every device. As the result is a maximal matching, tasks with high computation requirements are also matched, but if these two tasks would be put together, it is possible that no device in the network can run it. This is solved by keeping only the matches whose connected tasks can still run on a device within the most constrained 20% of the devices. An example is given with Fig. 3, where the two tasks with the highest bandwidth/weight get matched.

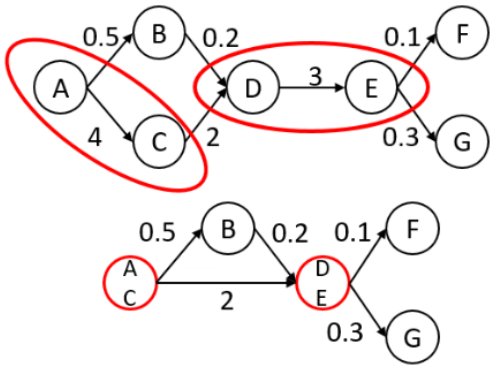

Fig. 3. Task Matching Example where largest weights get matched

C. Migration policy

Solutions do not stay valid, as the network and software requirements change over time. For this, the coordinator needs additional measures if it is to support dynamic networks. Multiple scenarios require software migration, but they all differ in software migration requirements. We define two migration types: required and optional. Required migration contains tasks which must be moved, whereas optional migration exists of tasks which are moved to optimize the global placement, but the requirements are not necessary.

Required:

- Initial placement: Unless all devices contain all tasks, tasks need to be moved from the task storage to the device which will run it.
- Agent failure: If an agent or its device fails, the tasks must move from the task storage to the new device. This occurs when a device loses power or leaves the network. Only that agent's tasks need to be moved.
- Agent shutdown: Graceful shutdown of an agent requires tasks to move from the agent it was running on to the new agent. Might happen when a device gracefully shuts down. Only that agent's tasks need to be moved.

Optional:

- Context change: If network links or device loads change due to external influences, more optimal placements can appear. Agent shutdowns and failures greatly influence this. Potentially all tasks are moved.
- New agent: If a new agent joins the network, tasks can be distributed to it which might cause the application to work better. Potentially all tasks are moved.

Note that context change migrations change from optional to required if constraints are no longer met.

D. Routing

For our results, we used Dijkstra to find the shortest path between two machines to measure the cost of placing the communication link over the path. This however does not match with realistic scenarios, as the most common routing protocols determine paths based on specific metrics, such as OSPF which focuses on bandwidth. These, however, can be implemented without too much effort.

V. Results

In this results section, we will compare the used approaches toward solving our task placement problem. This coordinator has been written in Python, using the JMetalPy framework for the implementation and algorithm fine-tuning [22].

A. Multi-Objective Optimization

We defined the user preference weights as a ranking, ranking latency first, WCET second, bandwidth third and energy consumption fourth. The following results will be compared using different metrics. As we are solving MOO problems, the concept of dominance is used to compare two solutions. A solution dominates another solution if every objective is

at least as good, and at least one is better. The dominating solution is thus considered better as the dominated solution. However, domination does not account for the full picture, as we are working with a DM, which has his solution preferences. This can result in a solution with a good bandwidth usage and energy cost, but a high latency, when the weights might indicate that latency is the most important. To solve this, we compare solutions based on preferences, using Eq. 2. This is used with normalized objectives and allows us to compare two different placements conform to the weights, where a lower cost is better according to the preferences.

B. Use Case

We will build on the use case proposed in [3], which lies in the automotive sector. Vulnerable road user safety, such as bikers and pedestrians, is improved by applying cooperative road user detection near crossings. This requires the combination of vehicle data and structural sensors, such as cameras. If any sensor detects an obstacle or road user, an alert will warn both the driver and the vulnerable road user of the issue. Moreover, the structural sensors can see more than the car sensors. This can be used to do predictions on the car's path and the vulnerable road user's path, and predicting or preventing potential accidents. An example network is shown in Fig. 4, over which we place a task graph of 15 tasks.

Table IV shows a technique comparison, where an average result was taken from 100 runs. As the exhaustive search is a deterministic algorithm, it was only calculated once. We clearly see that the FC-MOPSO algorithm tends to outperform both the GA as the NSGA-II algorithm based on the latency metric. It does make a trade-off at run-time, taking twice the amount of time. This is caused by the added evaluation complexity of FC-MOPSO. When considering the lowest ranked metric, namely energy consumption, the GA and the NSGA-II algorithms perform considerably worse. This can be attributed to the search space complexity, including constraints. If we reduce this by matching, this is no longer an issue. When looking at the run-time, it is shown that, due to the reduced complexity, the algorithms with a graph matching preprocessor do slightly better. However, the average time for calculating a matching on the use case is 0.25 seconds, which makes them perform slightly worse. Do note that due to the numerous constraints, exhaustive search performs well in this use case, using a parallel fail-early technique. Of the 9^{11} possible placements in an unconstrained scenario (9 free tasks over 11 devices), 23 million did not violate the constraints, resulting in about 0.07% viable placements.

C. Algorithm Comparison

The algorithm behaviour is visualised in Fig. 5. The relative frequency is the frequency of a solution over the total solutions, and is shown per algorithm over 100 runs. The FC-MOPSO shows his strength due to the intrinsic capability of handling constraints. The Genetic Algorithm (GA) is the worst performer. Due to the single objective optimization, it is unable to consider his priorities properly. Do keep in mind that the heuristics offer no solution quality guarantees. Fig. 6 shows us the algorithm's behaviour with a matched graph. Due to the reduced search space, the algorithms start to converge. Although Fig. 5 and Fig. 6 are kept separate for readability, we wish to point out that although the best solution that is found in the matched environment is considerably worse, the algorithms still outperform the regular algorithms. This search space reduction allows the algorithms to converge considerably faster, and tends to remove a large part of the constrained search space. Due to this reduction, dynamic networks suddenly become considerably more achieveable, as the exhaustive search can join competitively. Additonally, the graph matching stays the same as long as the application does not change, further reducing the search time.

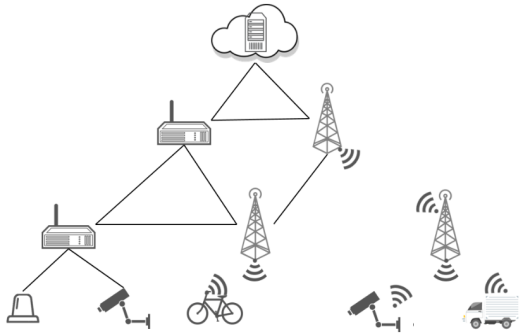

Fig. 4. Use Case

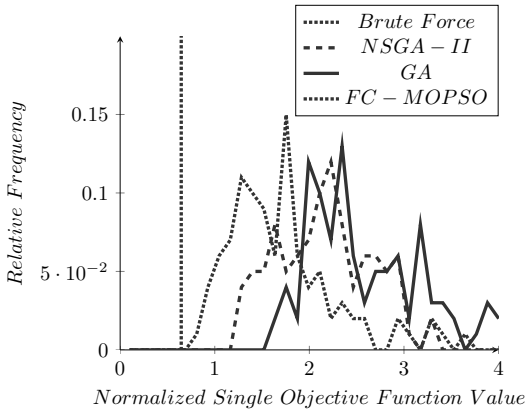

Fig. 5. Use Case Solution quality distribution

VI. Conclusion

In this paper, we defined the complexities of MOO on the task placement problem, and compared it with SOO. We show that in this constrained environment, the FC-MOPSO algorithm excels at finding solutions. When the constraints are reduced, other algorithms are able to perform as well. We have also shown under which circumstances graph matching is possible, and their improvement on the search. This improves the viability of placement over networks with static applications. This contribution can be applied to all placement problems, aiding in further optimization.

TABLE IV
COMPARISON OF AVERAGE OBJECTIVE RESULTS

	Latency (ms)	WCET (s)	Used Bandwidth (MB/s)	Energy Consumption (W)	Search Time (s)
Exhaustive Search	35	1185	1300	804.4	2349.345
Exhaustive Search-M	39	1392	1680	837.8	1.427
NSGA-II	68	1119	1820	1446.758	0.775
NSGA-M	56	1412	1680	837.496	0.672
GA	79	1332	2110	1308.748	0.741
GA-M	71	1335	2250	760.842	0.708
FC-MOPSO	58	1249	2520	899	1.596
FC-MOPSO-M	40	1416	2050	839.6	1.478

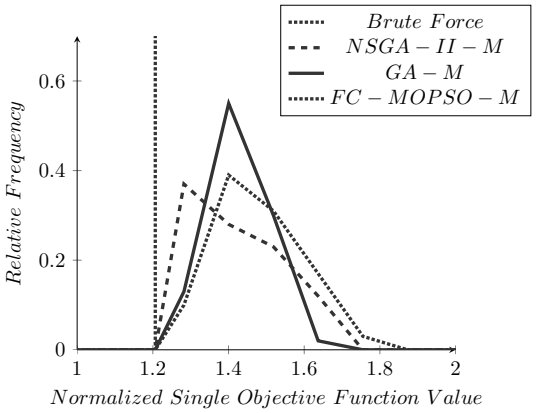

Fig. 6. Solution quality distribution for matched graphs on use case

VII. FUTURE WORK

As we enter more dynamic networks, task redundancy should be reviewed to cover for tasks which are on devices which leave the network. However, as these networks are often resource-constrained, implementing full redundancy might not be feasible. An alternative here is failure detection, recovery, prediction and prevention. An example is predictive maintenance, where hardware is continuously monitored, allowing the creation of a model of when the hardware is likely to fail, enabling the replacement before the failure. The complexity of failure recovery rises considerably when stateful software is ran on the devices, generating issues such as state storage and recovery. As we are working with a discrete problem, issues might arise when trying to locate the pareto front. This front can be non-continuous, requiring advanced MOO techniques. More research is required to tweak the algorithm and parameters so that improved results are achieved. Finally, attention should be brought to the security aspect of coordination. Attackers might grab control of the coordinator to force it to overload certain machines, causing potential failures. Alternatively, control of the coordinator allows injecting malicious software into the network, becoming a botnet orchestrator.

REFERENCES

[1] Cisco, "Cisco Global Cloud Index : Forecast and Methodology 2014-2019 (white paper)," *Cisco*, 2016.

[2] ——, "Cisco IOx Data Sheet," pp. 1–3, 2016.

[3] R. Eyckerman *et al.*, "Distributed Task Placement in the Fog: A Positioning Paper," Tech. Rep.

[4] A. Brogi *et al.*, "QoS-aware deployment of IoT applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1–8, oct 2017.

[5] E. D. Coninck *et al.*, "Middleware Platform for Distributed Applications Incorporating Robots, Sensors and the Cloud," *Proceedings - 2016 5th IEEE International Conference on Cloud Networking*, pp. 218–223, 2016.

[6] S. Vanneste *et al.*, "Distributed Uniform Streaming Framework: Towards an Elastic Fog Computing Platform for Event Stream Processing: Proceedings of the 13th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing," pp. 426–436, 2019.

[7] R. Eyckerman *et al.*, "Requirements for distributed task placement in the fog," *Internet of Things*, p. 100237, Jun. 2020.

[8] ——, "Context-Aware Distribution In Constrained IoT Environments," in *Proceedings of the 13th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 2019, pp. 437–446.

[9] Z. Tang *et al.*, "Migration Modeling and Learning Algorithms for Containers in Fog Computing," *IEEE Transactions on Services Computing*, vol. 14, no. 8, 2018.

[10] K. Deb *et al.*, "A fast and elitist multi-objective genetic algorithm:NSGAII," vol. 6, no. 2, pp. 182–197, 2002.

[11] V. Mokarram and M. R. Banan, "A new PSO-based algorithm for multi-objective optimization with continuous and discrete design variables," *Structural and Multidisciplinary Optimization*, vol. 57, no. 2, pp. 509–533, Feb. 2018.

[12] T. Huybrechts *et al.*, "A New Hybrid Approach on WCET Analysis for Real-Time Systems Using Machine Learning," no. 5, pp. 1–5, 2018.

[13] K. Krommydas *et al.*, "OpenDwarfs: Characterization of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures," *Journal of Signal Processing Systems*, vol. 85, no. 3, pp. 373–392, 2016.

[14] C. Liu *et al.*, "Data quality and the Internet of Things," *Computing*, 2019.

[15] M. Baldauf *et al.*, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, p. 263, 2007.

[16] Y. Xia *et al.*, "Combining Heuristics to Optimize and Scale the Placement of IoT Applications in the Fog," *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pp. 153–163, 2018.

[17] K. Deb, "An Estimation of Nadir Objective Vector Using a Hybrid Evolutionary-cum- Local-Search Procedure An Estimation of Nadir Objective Vector," pp. W–470, 2009.

[18] R. Marler *et al.*, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, apr 2004.

[19] E. Lansing, *Investigating the Normalization Procedure of NSGA-III*. Springer International Publishing, 2019, no. May.

[20] K. Deb *et al.*, "Towards estimating nadir objective vector using evolutionary approaches," *GECCO 2006 - Genetic and Evolutionary Computation Conference*, vol. 1, pp. 643–650, 2006.

[21] S. Wang *et al.*, "Integrating Weight Assignment Strategies With NSGA-II for Supporting User Preference Multiobjective Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 378–393, 2018.

[22] A. Benítez-Hidalgo *et al.*, "jmetalpy: A python framework for multi-objective optimization with metaheuristics," *Swarm and Evolutionary Computation*, p. 100598, 2019.