Requirements for distributed task placement in the fog

Reinout Eyckerman reinout.eyckerman@uantwerpen.be
Siegfried Mercelis siegfried.mercelis@uantwerpen.be
Johann Marquez-Barja johann.marquez-barja@uantwerpen.be
Peter Hellinckx peter.hellinckx@uantwerpen.be
University of Antwerp - imec, IDLab - Faculty of Applied Engineering, Sint-Pietersvliet 7, 2000 Antwerp, Belgium [1]

1 Introduction

Using Internet of Things (IoT), we are starting to gain better knowledge and control of our environment. The added sensors and actuators allow both consumers and industry to automate, control and monitor their surroundings. The industry sector is utilising IoT for the next industrial revolution. IoT is leveraged to monitor processes and equipment, which enables predictive maintenance and prevents downtime costs. The automotive sector benefits greatly as well. The upcoming smart vehicles, which use photodetectors, rain sensors, cameras, and various other connected sensors and actuators to create one large connected system. To expand their electronic horizon, smart vehicles can be connected together to create Vehicle Ad-Hoc Networks (VANETs), allowing vehicles to communicate their information to each other. Vinitsky et al. [1] found that only a 10% smart vehicle penetration rate is enough to reduce traffic congestion by 25%. This shows us that only a few smart vehicles can already make a difference.

However, with the increasing adoption of IoT, a considerable amount of data is generated, putting a serious strain the network. Cisco estimates that by 2021, 850 ZB of data will be generated yearly, more than triple the amount since 2016 [2]. Most of this data is, however, short-lived: Cisco estimates that 90% of the data will be used for processing and will not be stored. Data generated by IoT devices have especially short life-spans, as it mainly needs to be processed and acted upon. Using Fog computing, the state of the art attempts to enable pervasive access to a shared set of computing resources. This creates a platform for distributed and latency-aware applications. The application tasks will be placed in this fog, an intermediate network layer, shown in Fig. 1, allowing for a reduced network load and latency. However, fog networks often contain a highly dynamic aspect. This can be seen in the previously mentioned VANETs, where vehicles continuously (dis)connect due to passing by roadside access points, or differences in speed.

To ensure a minimal latency and bandwidth usage, a task placement coordination technique is

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

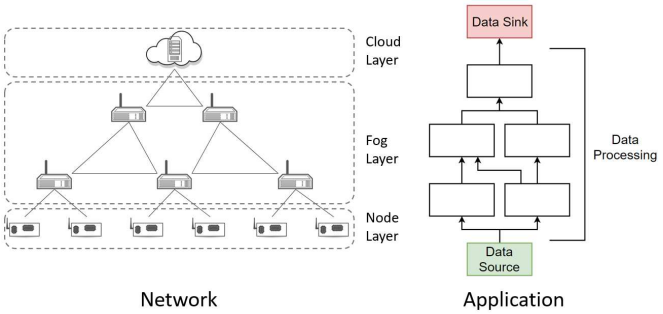

Figure 1: Example of a fog network & an application graph.

required. This counts especially for highly dynamic networks so as not to lose application efficiency when nodes move/disappear. Applications must first be divided into separate tasks, which can be distributed over the network. The coordination technique then keeps in mind the available hardware and network resources along with the dynamic network aspect, and distributes these tasks over the network. Depending on the scenario, the potential overhead and communication cost can make it infeasible to distribute tasks in a centralized manner. Thus, a distributed task placement approach is proposed, which adapts the task placement to changes in context. We define such a context as a collection of Key Performance Indicators (KPI), which represent device and metric-specific weights. The KPI con change depending on the significance of a hardware resource (e.g. energy scarcity when running on battery power).

This paper entails possible approaches for efficient task placement across the network. This approach can enable application latency reductions, increased application efficiency, automated placement, so that placement is no longer the concern of the application developer. The research presented in this paper focuses on fog computing, and the issues when distributing tasks to enable this computing paradigm. We define the difficulties provided by the task placement problem, and propose a distributed algorithm. We then simulate this algorithm and validate it against several other Single-Objective Optimization techniques.

2 State of the Art

Many challenges appear when attempting task placement on fog networks, as presented by Wen et al. [3]. They described several challenges, such as the IoT device heterogeneity, security, network latency, dynamic behaviour, and fault tolerance. It becomes clear that the problem spans multiple research areas, which we will describe below.

2.1 Hardware & Software Models

Before being able to place tasks on devices, we need models of both the tasks and the devices. Device metrics encompass CPU, RAM and disk capacity, among others, as defined by Xia et al. [4]. As fog devices are heterogeneous, we need to define comparable metrics which can be used across devices. Huybrechts et al. [5] provide us with a look into the application model, focusing on Worst Case Execution Time (WCET) analysis using a hybrid approach. This goal was achieved by using the COBRA and TACLeBench tool suites [6]. They were able to efficiently calculate and model WCET using this approach.

2.2 Constraints

Finding the optimal software placement is a constrained problem, as links for example only have a certain bandwidth capacity and devices only have a certain amount of available memory. Xia et al.[4] considered the constraint where the device has a set of available resources, such as memory, whose usage cannot be exceeded. Similarly, Sharma et al. [7] considered constraints coming from the application. Software heterogeneity was considered while placing tasks on Google compute clusters. One of such constraints is the requirement of a minimum kernel version to properly run. They then measured the performance impact of such constraints on the task scheduler.

2.3 Distributed Software Frameworks

The development of distributed software provides more challenges than monolithic software, such as message passing and state maintenance. Vanneste et al. [8] developed the Distributed Uniform STreaming (DUST) framework as a solution to such challenges This middleware provides a transparent communication layer on top of which distributed applications can be built. Removing the complexity of communication while keeping a high amount of configurability helps to cater both new and experienced distributed software developers. Developers build parts of applications on these blocks, creating a connected distributed application, which can, in turn, be distributed over the network.

2.4 Context

The tasks run on devices which are in a certain context. The concept of context-awareness has been around for a long time. As it encompasses many areas, many have tried to define what exactly a context is. One popular definition was coined by Dey & Abowd, whose definition included user, application, location, and device awareness [9]. Although all these contexts do influence task placement, only device awareness influences it directly. Location and user awareness could have an influence on the application. This could, however, be simplified into a software requirement change. One example is that if it gets dark outside, regular cameras will not be of much use on smart vehicles and can thus be disabled. As the amount of data the task has to move is lowered, the requirements are changed.

2.5 Problem Solving Techniques

There are several approaches for solving the task placement problem. One is by use of metaheuristics. Xia et al. [4] looked into application placement on a fog network. They used dedicated zones as application deployment areas to ensure placed tasks their locality. This is used when placing his tasks over his devices using several solutions, such as exhaustive search, naive search, and improvements of naive search, optimizing the weighted latency. Wang et al. [10] looked into optimization techniques in the edge computing paradigm. Although different from fog computing, edge and fog both attempt to reduce the network load by moving applications closer to the source. They proposed an Multi-Access Edge Computing (MEC) application placement algorithm, a technique where they placed micro-clouds closer to the edge. Wang et al. then compared their technique to a greedy and the vineyard algorithm. Their research used Linear Programming methods for mapping tree application graphs onto tree network graphs. In previous research we investigated generalized methodologies such as a brute-force and genetic algorithms to solve tree-based graphs [11]. If we want to solve the placement problem in a distributed fashion, additional large complexities are added. One distributed approach is the Contract Net Protocol, as defined by Smith [12]. There, each device contained an agent, who, using a simple bidding mechanism, placed the tasks one by one. Another approach for distributed optimization is the A-Team, as defined by Talukdar et al. [13]. They defined a set of autonomous agents, called the Asynchronous Team (A-Team), which cooperate to solve the same goal. Barbucha & Je [14] then validated this approach by trying to solve the Vehicle Routing Problem. Barbucha also showed the efficiency of distributed agent optimization [15]. Teams of agents were placed on a distributed network which showed that an increase of agents improves efficiency up to a certain point, after which the overhead of an additional agent becomes too large to increase the efficiency further. Andrzejak proposed several techniques for distributed

organization as well [16]. One of these were ants traversing the network, leaving pheromones where suitable placements were provided.

2.6 Simulation

To simulate the coordinator and accurately estimate the emergent behaviour of distributed coordinators, simulation can be used. One versatile simulator for distributed coordination simulation is Simgrid, which has been actively developed since 2001 [17]. This simulator allows concurrent software stack simulation on top of simulated hardware resources, and can optionally use ns-3 for network simulation. Simgrid allows modeling the application and tests its behavior, speeding up research. Other simulators exist, most focusing on the specific behavior of IoT networks and their models such as IoTSim [18], which attempts to mimic the behavior of an existing application to generate results about certain metrics and its behavior in the network.
This State of the Art covers all the fundamental parts required for developing a task placement coordinator. We will define several problems not stated in the state of the art.

3 Problem Definition

Our main objective is to show the current problems involved when trying to distribute tasks across IoT/Fog networks using both centralized and decentralized optimization algorithms. A centralized approach has several drawbacks. It might be too slow for time-critical applications (communication or optimization might take too long) The monitoring of global resource availability by a centralized entity in real-time introduces plenty overhead. Additionally, in a highly dynamic environment a single point of failure is quite dangerous, as the failure of a single device causes the failure of the entire software stack. On the other hand, using a decentralized approach might take too much resources of the devices, blocking the placement of several tasks on a single device. A global optimum is not ensured, and will most likely get stuck in local optima. The communication of the agents on the devices creates an additional network overhead. We will now define the metrics we use for our network and application. After this, we define our context and the Multi-Objective Optimization (MOO) problem. Finally, we describe the features an efficient problem solver should have, and a use case.

3.1 Metrics

Metrics for both the software application and the network hardware must be defined. In Table 1 we show the selected network metrics, and in Table 2 the selected application metrics. The chosen application metrics can almost directly be mapped to hardware metrics, with the exception of the worst case network load, which can be mapped to both the bandwidth of the network links as to the Network Interface Card (NIC) transfer speed of the device. Various other metrics exist, such as software constraints (availability of a certain OS), hardware constraints (availability of a certain sensor) and network specifics (such as wireless interference). The network model can be created by using the hardware characteristics. A model of the application can be created using the COBRA Framework [6].

Table 1: Network Metrics

Name	Description
N	Set of physical nodes in the network
L	Set of physical links in the network
n_cpu	Processing Speed of node n
n_mem	Memory Size of node n
n_nic	NIC Transfer Speed of node n
n_ec	Energy Consumption of node n
l_bw	Bandwidth (kbps) of link l
l_lat	Latency of link l

Table 2: Software Metrics

Name	Description
A	Set of tasks in the application
C	Set of data links in the application
a_wcet	Worst-Case Execution Time of task a
a_wcmc	Worst-Case Memory Consumption of task a
a_wcec	Worst-Case Energy Consumption of task a
c_wcnl	Worst-Case Network Load of data link c
c_mlat	Maximum Allowed Latency of data link c

3.2 Network & Application Shape

The network and the application can both be modeled as a graph. Previous research often used tree graphs for the network, and directed linear [11] or tree graph structures for the application [10]. Constraining the graphs to a tree or a linear graph allows for constraint modeling into the optimization techniques. In order to improve realism, however, we chose to use directed multigraphs. Mesh-like networks are underrepresented due to them greatly enlarging the search space. They are however necessary to exploit the full benefits of the network, such as network loops and redundancy. If we then make these mesh-like networks directed, we can easily define different properties for up and downlinks. ISPs for example often throttle uplink speed, which is easily modeled in a directed network. Similarly, we use directed multigraph applications, in order to support any kind of application. Example graphs are shown in Fig. 1.

3.3 Context

To accurately model device requirements apart from their available resources, we define a device-specific context. This context is a set of KPI, each connected to their respective device parameter. Using these KPI, the device can tune itself with regards to task placement For example, the energy KPI will go up if the device has a limited amount of energy, stopping energy-hungry tasks to be assigned to the device.

3.4 Multi-Objective Optimization

Table 3: Objective Functions

Objective	**Function**
Energy	$\sum_{n}^{N}\sum_{t}^{n} n_{t,e} * n_{ec}$
Bandwidth	$\sum_{c}^{C}\sum_{l}^{c_{route}} l_{bw}$
Latency	$\sum_{c}^{C}\sum_{l}^{c_{route}} l_{lat}$

Due to the large amount of optimizable parameters, the optimal task placement is a Multi-Objective Optimization (MOO) problem. These are generally difficult problems to solve as there are multiple, potentially conflicting, criteria are to be optimized simultaneously using multiple objective functions, e.g. optimizing toward energy efficiency and maximizing machine utilization. Due to a large set of device constraints, the complexity tends to increase further. Reducing these constraints reduces the difficulty of solving the problem, but also decreases the accuracy of the models. When solving multi-objective problems, the result is a set of optimal solutions, called the Pareto front, where no criteria can be improved without degrading other criteria. For more information around pareto-optimality in the context of engineering, we redirect the interested reader to [19]. Several objective functions are shown in Table. 3. This is a non-exhaustive list, as minimizing the Worst Case Execution Time (WCET) or usage cost could be considered as well, and depend on what your use-case requires. Marler & Arora [19] provide a survey paper about available techniques for solving MOO problems. There are three main techniques for working with MOOs.

- A priori: a subset of Pareto-optimal solutions is found by modeling preferences in advance.
- A posteriori: the entire Pareto-optimal set is determined, after which the user selects a solution based on personal preferences.
- Interactive method: the user iterates over the problem multiple times, allowing him to fine-tune his preferences.

We want a completely autonomous coordination technique, where applications are placed without human interaction or feedback. However, autonomous placement can only occur when coordinator is able to find a single optimal solution, since finding multiple equally good placements is something it cannot work with because it would not know which solution to follow. Preferences need to be known in advance to do this. It is the system administrator's task to determine this, as it is use-case dependent. In this regard, the only suitable approach is the *a priori* method. This is the only method that can work without user interaction. If this is then modeled into a Single-Objective Optimization function, only a single value is optimized, allowing the coordinator to find a single best within the pareto front. Using the single-objective approach within the search algorithm itself can create problems in finding Pareto-optima, depending on how well the function is constructed. Two *a priori* methods are defined below:

3.4.1 Weighted Sum:

The weighted sum approach is a simple yet widely used approach, where the set of objectives are scalarized into a single objective function. This is done by multiplying each objective function with a user-defined weight. Some issues can occur when using the weighted sum method, as defined by Marler & Arora [20]. A major issue is objective function normalization, where the ranges of the objectives need to be known before they can be normalized. Normalization does simplify the weight defining process, since due to objective function normalization the weights no longer need to take in account the scale of the objective function.

3.4.2 Lexicographic Method:

When using the lexicographic method, the objective functions are sorted and solved in order of importance. This does not create a single quality of the solution, but allows the comparing of

two solutions to find the better one. Kaufman & Michalski [21] provided multiple techniques to aidi prioritization of objective functions. A technique was presented where multiple parameters are defined per objective function. These correspond to the results they present in the set of Pareto solutions.

3.5 Optimization Technique Features

The features have an impact on the technique's efficiency for the application placement problem and can be used as measurements. These features will be defined next.

3.5.1 Search Space Knowledge

This item will impact all of the other features. If the optimization technique does not know the entire search space, several issues might arise, as defined next.

Normalization The technique should be able to work with either un-normalized objectives or have the capability to normalize the objectives. This is required to properly work with the single-objective functions. The weighted sum for example, cannot work with un-normalized data, as defined before. Large-valued objectives would dominate the cost with smaller values barely influencing the end result. However, normalizing the data requires knowledge of the range in which the objectives will fall. This range can be estimated if the entire search space is known, but this becomes a severe challenge when taking into account partial search space knowledge, which occurs a lot in distributed optimization. Depending on the network there might be more bandwidth or more latency, which requires calculating the ranges for each network specifically, blocking any general approaches. This issue can be solved by using for example the lexicographic single objective method. This however only allows the comparing of multiple solutions, and does not provide a quality measure of the solution, which is required by several meta-heuristics.

State Formulation Algorithms often use a complete-state formulation, allowing the exact calculation of the quality using for example the weighted sum method. However, this formulation becomes infeasible without knowledge of a large part of the search space, as there is not enough information to create a complete state. The alternative here is the incremental formulation, where the solution is built step by step. The issue here is that it becomes difficult to compare incremental solutions correctly, as the future development of the incremental solutions can greatly influence the quality. Using a heuristic can help solve this problem. However, creating admissible or consistent heuristics is a challenge, as often a lot of search space information is incorporated into the heuristic.

3.5.2 Distribution

Removing a Single Point Of Failure (SPOF) greatly increases the reliability, but also the complexity. Communication overhead should be minimal so the network links are not overloaded. Due to the large heterogeinity, a distributed algorithm should be able to run competitively on smaller devices so that they can add to finding the solution. If this is not the case, stronger devices might have to wait on solutions of the smaller devices. Such an algorithm also requires a certain fault tolerance so that a solution can still be found when a device misbehaves/fails.

3.5.3 Speed

The speed of the used technique is another relevant feature. As fog networks are highly dynamic, both on context as on device level, solutions can quickly lose their value. If the algorithm is unable to find a new solution quickly, distributed tasks might perform badly or devices might get overloaded with tasks.

3.5.4 Resource Consumption

The resource consumption especially important in distributed algorithms. If the technique has a large memory or processing power consumption, some devices might not be able to run any other tasks. This may result in highly unreliable devices in highly dynamic networks.

3.5.5 Scalability

This feature asks that the technique is scalable to larger networks and applications. Some networks can go into hundreds of devices, or hundreds of application tasks. The algorithm should still be able to find a solution while respecting the other features, such as speed or resource consumption.

3.5.6 Solution Memory

Solution memory can be another important aspect, for it can speed up sequential task placements, and might keep new optimal placements in the neighborhood of the existing placement, requiring little change in current task placement. This is especially useful in continously changing networks. As this could result in a potentially less optimal placement, this would also result in fewer application problems since most tasks can be kept where they are running.

3.5.7 Global/Local Optima

The final feature should test how well techniques which go for local optima (Contract Net Protocol, Hill Climb) compare to those looking for global optima (Genetic Algorithm (GA), Particle Swarm Optimization). As it is a heavily constrained search space, global optima might be surround by considerably worse placements, making it hard for local optimizers to find them.

3.6 Techniques

As stated in the state of the art, there are multiple techniques that can be used. Due to the design of the application and network graphs, there is a very large search space with worst-case $O(N^A)$ complexity, making exact search techniques often infeasible. Meta-heuristics can bypass this problem. However, finding and composing a good meta-heuristic is interesting yet complex research, and greatly depends on the choices made when selecting a MOO technique. The selected technique should be inspected for features which can be implemented in the meta-heuristics for pre-optimization, improving search efficiency. Another selection to be made is the multi-agent technique, such as Asynchronous Team (A-Team), combining it with meta-heuristics.

3.7 Simulation

The Simgrid simulator platform shows to be interesting for testing the approaches. However, there is a limit on realism in simulation. This can be increased by modeling set of VMs into the network to be simulated. These VMs can model the resources available by the devices. Network simulation could be done using the Linux *netem* command, allowing the changing of link bandwidth and latency. Using this VM technique, the effect of both the distribution technique as that of the application distributed can be observed. Realism can be further increased by using a test-bed, using, for example, the Fed4Fire test-bed [22]. The use of a test-bed, however, is time-consuming, and should thus be well-prepared by simulation before actual deployment, as finding errors becomes quite complex on an actual test-bed.

3.8 Use Case

We find our use case is found in the automotive sector. Implementing cooperative detection of vulnerable road users near a crossing can increase safety. This is done by combining vehicle data with infrastructure sensors. The network is shown in Fig. 2. It describes infrastructure cameras monitoring for vulnerable road users, and a roadside unit connected with a vehicle, with vehicular proximity sensors for additional detection and a graphical interface for the driver. Potential issues are sent to the driver's display. The application preprocesses the data before reaching the cloud server, where it accounts for data checking. Afterward, data compression and conversion happens before finally reaching the target device. The application graph is shown in Fig. 3.

4 Distributed Reconnaissance Ant Colony Optimization

We took inspiration from Andrzejak et al. [16] for a distributed algorithm implementation. They described a methodology based on Ant Colony Optimization (ACO), where the ants traverse the actual network, where the pheromones describe how well a task would run on a specific device. These ants then report their found placements back to the service manager, who then determines where the tasks will be placed. We expanded on this research, tackling unhandled issues and enabling multi-objective optimization. Our algorithm has been coined Distributed Reconnaissance Ant Colony Optimization (DRACO), as Distributed ACO is focused on distributing the algorithm, whereas we focused on actual ants discovering the network instead of simulated ants. Due to the low complexity of the ants, there is a minimal resource usage of the devices. We will describe the components of the algorithm below.

4.1 Service Manager

The service manager is responsible for selecting the optimal placement for a certain application. It creates the ant colonies and ensures an optimal placement is selected. Although one might consider this manager a SPOF, it actually makes sense to have a manager responsible for the application, as it is the device that requires the application to run (e.g. data should not be processed if there is nothing to store or use it).

4.2 Ant Colony

The colonies are placed across devices, and send out ants. For this research, we chose to place a colony on every device that has a task that is only allowed to run on said device. As the application graph is a directed multigraph, there might be tasks locked to devices in the middle of the application graph. Having a widespread distribution of colonies allows more accurate pheromone distribution, as ants are only allowed to place tasks that are neighboring already placed tasks.

4.3 Server Manager

A server manager manages the ants and the pheromones for the device on which it is running. It stores the pheromones produced by an ant in a pheromone table, and provides device and link capabilities and metrics to the ants. Once every while, a server manager distributes his pheromone table to neighboring devices. This allows the spreading of knowledge to other devices, allowing ants to know where to head next.

4.4 Ant

The ants get created by their respective colony and traverse the network. On each device it determines the quality of running certain tasks on the specified device. It determines the quality of tasks that are directly connected to tasks it has already placed across the network. This ensures that the quality function of a device integrates at least a part of the used network metrics. Once it has calculated the quality for the current device, it will place the task on the current device, except for two scenarios:

- The quality value of a task is unknown on a neighbor: The ant will jump to said neighbor, calculate the quality and place the task there. Forcing ants to place it on a neighbor with no known quality helps ants exploring the network in the initial stage of the algorithm.
- The quality value of a task is better on a neighbor: The ant will jump to said neighbor, re-calculate the quality and place the task there. This one hop enables exploration of the algorithm, while still forcing exploitation of the pheromones.

This means that the ant can skip a single hop before being forced to place a task and continue the search. The ant determines the placement quality using the function defined in the next section.

4.5 Quality Function

To determine if a task fits well on a certain device, it determines the cost of the tasks placed upon it according to the weighted sum method. The objectives which are related to the network are slightly manipulated however. We solely calculate the cost of placing the task on the device and the communication costs it has with the other directly connected tasks, normalized. If we take into account the placement of the other previously placed tasks, the influence they have on the network objectives would not be representative for the other ants which are working with differently placed tasks.

$$obj_{bw} = bw_{used}/bw_{minreq} \tag{1}$$

Here, the bandwidth objective is calculated by searching how much bandwidth the tasks placed on the device use, and dividing it by the minimal required bandwidth if tasks were one hop away.

Note that the minimum requirement includes the communication links referring to yet unplaced tasks. Adding these allows ants to take into account the potential effect of unplaced tasks. If the pheromone table already contains information about the quality of the task, it will update this value with the newly computed value according to the following function, based on [16]:

$$p' = \gamma \cdot p + (1 - \gamma) \cdot r \tag{2}$$

Here, γ represents the relevance of the historical quality in the range [0,1], and is usually set to a large value. p represents the current quality, and r the original quality factor. An ant selects which task to place based on how well the calculated quality factor is. If the constraints are not respected on a device, the quality factor is set very large. Tasks are not placed if they exceed the defined constraints. This can generate scenarios where the ant traverses the entire network in search to place a task that cannot be placed anywhere, due to the previous placement. The pheromone tables try to prevent this. If this does occur, the ant will eventually terminate after a predefined amount of hops.

4.6 Review

We will now compare this newly-defined algorithm to our previously defined features. The search space knowledge is partial, as the service manager does not know the entire network. Because of this, an incremental state formulation was chosen. The difficulties of normalization are left open for future research. For now it is assumed that the system administrator can provide sane normalisation ranges. As the only single point of failure for the algorithm is the service manager, it can be considered fully distributed. This single point of failure is not an issue, as it is considered the service in need of the placed application. As the ants only depend on the device they are on, speed should also not be an issue. Due to the simplicity of the ants, the device resource consumption should be kept to a minimum as well. Although there are quite some ants traversing the network, the size of an ant is only as large as its currently found placement, creating a minimal impact on the network. The sharing of the pheromone tables can create more load, but this can be reduced by decreasing the update rate of the pheromone table. The scalability of the algorithm can be a considerable issue. As a colony is launched on every device with a locked task, there might be a flood of ants on the network. This might however not be an issue, as the amount of unlocked tasks might be considerably lower, decreasing the search space and thus the lifetime of an ant. The pheromone trails can be used as solution memory. However, on networks with large changes these pheromones can become incorrect and actually throw off ants. As the complete search state is unknown, only local optima can be achieved.

5 Validation

We have provided a small test scenario to validate our results. In this scenario, we test our coordination techniques and compare them. All the compared algorithms are single-objective optimization algorithms. This allows the comparing of the DRACO to other algorithms which do not exploit the Pareto front. A simplified scenario is provided, which differentiates from future research since currently, placement calculation happens centralized.
The use case, which was previously defined, is shown in Fig. 2. The network is a static network, purely for testing placement techniques. Four different contexts are used, where the cloud, edge

devices, actuators, and monitoring devices have a different context. The colored nodes represent locked tasks, which are forced to run on a certain device, and thus cannot be moved to different devices.. They represent sensors, actuators and data validators. We normalize the objectives using the NSGA-II algorithm with an adapted crowding distance in order to find the edges for the Pareto front [23]. The single objective function we use is the following, based on the weighted sum method [11]. All algorithms were built using the JMetalPy framework [24].

$$C = \sum_{i=0}^{\#Components} \sum_{j=0}^{\#KPI} w_{ij} C_{ij} \tag{3}$$

Here we try to minimize the placement cost of component i onto the device with Key Performance Indicators (KPI) cost j per device. Sum these costs over the entire network for the global cost. Several techniques are used to solve this problem, briefly listed below.

5.0.1 Branch & Bound

A baseline is provided by a branch & bound with fail-early constraint checking. It is guaranteed to return the best placements, by iteratively checking all possible placements. However, since it has to check all possible placements, it has to calculate the objectives of $O(N^A)$ solutions. However, as an incremental state is used for the fail-early constraint checking, the constraints are calculated considerably more often than the objectives themselves. This makes is infeasible to use this technique in practice on large networks.

5.0.2 Hill Climb

The Hill-Climb (HC) algorithm is used as a local optimization algorithm. It is based on a multiple restart steepest descent HC, configured for 10 restarts. A greedy approach is used to improve total cost by moving the task to a neighboring device that benefits the solution the most As this is a local optimization algorithm, it has a high possibility to get stuck in local optima which might be invalid, constrained solutions. Due to this, it will have a hard time in highly constrained solutions. This can be solved by ensuring that the randomly generated start placement is unconstrained. Finding an unconstrained placement might be very hard however, as it is possible that only a very small subset of the search space is unconstrained.

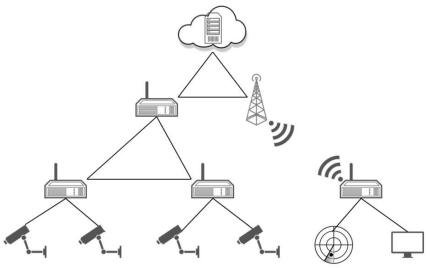

Figure 2: Use Case Network Graph

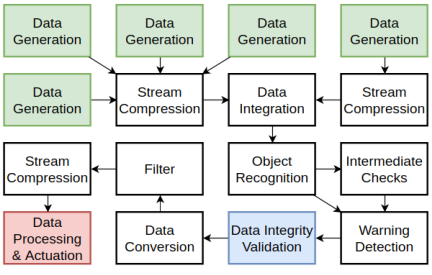

Figure 3: Use Case Application Graph

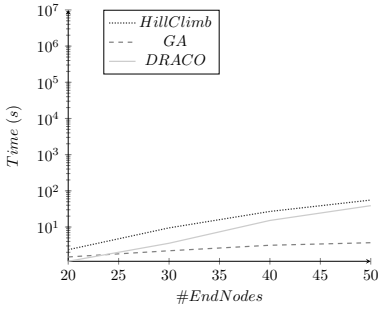

Figure 4: Algorithm Running Time with as much tasks as nodes

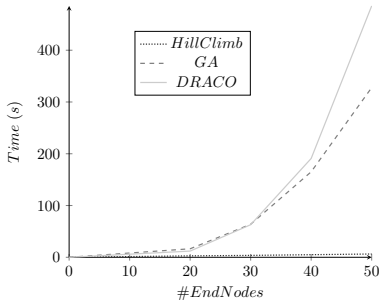

Figure 5: Algorithm Running Time with twice as much tasks as nodes

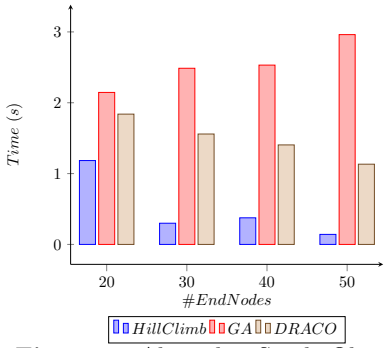

Figure 6: Algorithm Single-Objective Function Output

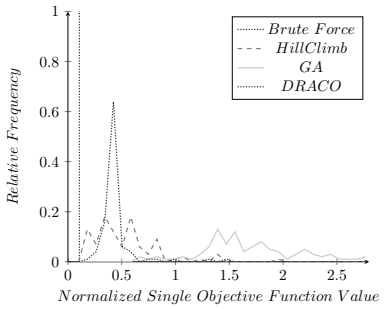

Figure 7: Comparison of calculated result

5.0.3 Genetic Algorithm

A general GA is used as a baseline for global optimization. We define a chromosome as a complete task placement. Ranking is based on the weight and is scaled according to the formula $1/\sqrt{n}$, where n is the normalized position in the ranking. The best children according to this ranking are selected and automatically pass to the next generation. Crossover happens using Simulated Binary Crossover. Parents for this function are selected using roulette-wheel selection. Using a polynomial mutation, solutions are mutated.

5.0.4 Distributed Ant Colony Optimization

Although the DRACO is inherently distributed, we simulate it to validate the results without having the complexity of implementing the actual agents. Although it is simulated, the limited knowledge is still incorporated, allowing accurate results and easy porting to an actual distributed environment.

5.1 Test Setup & Results

The used hardware is a desktop with a Intel Core i7-6700 CPU and 16GB memory.

Next to the use case, random networks were developed with an increasing amount of nodes. For

each of these networks, we generated two applications, where a subset of the application was locked to specific devices, preferring devices on the edge of the network as they simulate sensors/actuators. The first of these two applications has the same amount of tasks as there are nodes in the network, and the second one has the double this amount. Each algorithm was run ten times and then averaged. In Fig. 4, it is visible that all the algorithms scale quite well when the application graph does not grow too large. Even centralized, the DRACO outperforms the HC in terms of speed. However, Fig. 6 showcases the the trade-off we are making. Increasing the search space does not bother the GA much, which is to be expected as it is a global-optimization meta-heuristic. However, the quality of the GA results do not improve. The gap between the average quality of the GA and the DRACO and HC increases significantly. This is likely due to the large amount of objectives and constraints, something the genetic algorithm has issues dealing with. We notice that the DRACO does quite well compared to the GA, although it does take a considerable amount of time to run. We would like to stress to the reader here that the increase in calculation time of the DRACO is to be expected as the network size increases. Due to our simulation of this network, increasing the amount of devices increases the amount of simulations we have to execute. On a real system, this would not be the case as every device calculates it's own optimization functions, but as we are simulating these, these calculations are centralized. In Fig. 5 we notice that the DRACO and HC do not handle large applications well, with the GA being relatively unphased. As mentioned before, the simulations add to the cost of calculating the DRACO, a cost that falls away when implementing it in a distributed fashion. The HC is expected to exponentially increase in duration as well. Since the amount of tasks and devices increases, the amount of options that the HC has to consider explodes in number.

Finally, looking at the use case scenario in Fig. 7, we can see the probability of the result of each algorithm. With the Brute Force algorithm as baseline, we can see that the GA has a hard time getting close. This is likely suffering under the large amount of constraints. As expected, both the HC and the DRACO do quite well, although the DRACO outperforms the HC. This is due to the shape of the application. The placement of a colony on the cloud due to the cloud dependency a task has, allows for more accurate pheromones across the network.

6 Conclusion & Future Work

Fog computing will help in relieving network stress, but this needs a coordinator function to organize the fog processes. We defined the difficulties of coordination and developed a distributed algorithm to find optimal placements. It provides an approach and ends its case with a test scenario. From this scenario can be concluded that the HC tends to do better in regard to solution quality, but the GA provides faster results. Our proposed DRACO is suitable for smaller-scale scenarios, with large-scale scenarios requiring further validation. The time spent executing the DRACO algorithm on an actual testbed is considerably different, as we run a centralized simulation and as no network communication is taken into account. Additionally, this paper lists some of the most important literature in this field and defines some problems that are yet to be solved. A briefly touched subject in this paper is the problem of network monitoring, including network discovery and link monitoring. This is, however, a significant challenge when it comes to dynamic networks since it is infeasible to keep the network status up to date at all times. Another major problem is the current lack of security: if an attacker manages to get the coordinator to distribute his software, he gets an arsenal of resources to his disposal, or he can make the placement extremely inefficient by changing the weights. Load balancing should be implemented, with a detection mechanism that is able to

discover when a task is no longer able to process the stream due to too much information and can inform the coordinator about this so that an extra task can be deployed. Another interesting addition to add to this work is to add software constraints, as defined in [7].

References

[1] E. Vinitsky, K. Parvate, A. Kreidieh, C. Wu, A. Bayen, Lagrangian Control through Deep-RL : Applications to Bottleneck Decongestion, IEEE Intelligent Transportation Systems Conference (2018) 759–765doi:10.1109/ITSC.2018.8569615.

[2] Cisco, Cisco Global Cloud Index : Forecast and Methodology 2014-2019 (white paper), Cisco (2016) 2016–2021.

[3] Z. Wen, et al., Fog orchestration for internet of things services, IEEE Internet Computing 21 (2) (2017) 16–24. doi:10.1109/MIC.2017.36.

[4] 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) (2018) 153–163doi:10.1109/UCC.2018.00024.

[5] T. Huybrechts, S. Mercelis, P. Hellinckx, A New Hybrid Approach on WCET Analysis for Real-Time Systems Using Machine Learning (5) (2018) 1–5. doi:10.4230/OASIcs.WCET.2018.5.

[6] IDLab, Imec, University of Antwerp, COBRA Framework.
URL `http://cobra.idlab.uantwerpen.be/`

[7] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, C. R. Das, Modeling and synthesizing task placement constraints in Google compute clusters, in: Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC '11, ACM Press, 2011, pp. 1–14. doi:10.1145/2038916.2038919.

[8] 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 (2019) 426–436doi:10.1007/978-3-030-02607-$3_3$9.

[9] G. D. Abowd, et al., Towards a better understanding of context and context-awareness, Lecture Notes in Computer Science 1707 (1999) 304–307. arXiv:arXiv:1310.4038v1, doi:10.1007/3-540-48157-$5_2$9.

[10] S. Wang, M. Zafer, K. K. Leung, Online Placement of Multi-Component Applications in Edge Computing Environments, IEEE Access 5 (2017) 2514–2533. arXiv:1605.08023, doi:10.1109/ACCESS.2017.2665971.

[11] R. Eyckerman, M. Sharif, S. Mercelis, P. Hellinckx, Context-Aware Distribution In Constrained IoT Environments, 2019, pp. 437–446. doi:10.1007/978-3-030-02607-$3_4$0.

[12] R. G. Smith, Communication and Control in a Distributed Problem Solver, IEEE Transactions on Computers C (12) (1980) 1104–1113. doi:10.1109/TC.1980.1675516.

[13] S. Talukdar, L. Baerentzen, A. Gove, P. De Souza, Asynchronous Teams: Cooperation Schemes for Autonomous Agents, Journal of Heuristics 4 (4) (1998) 295–321. doi:10.1023/A:1009669824615.

[14] D. Barbucha, P. Je, An Agent-Based Approach to Vehicle Routing Problem, International Journal of Applied Mathematics and Computer Science 4.1 1 (2007) 36–41.

[15] D. Barbucha, Agent-Based Optimization, in: Agent-Based Optimization, Vol. 456, 2013, pp. 55–75. doi:10.1007/978-3-642-34097-0.

[16] A. Andrzejak, S. Graupner, V. Kotov, H. Trinks, Algorithms for self-organization and adaptive service placement in dynamic distributed systems, HP Laboratories Palo Alto.

[17] H. Casanova, A. Giersch, A. Legrand, M. Quinson, F. Suter, Versatile, scalable, and accurate simulation of distributed applications and platforms, Journal of Parallel and Distributed Computing 74 (10) (2014) 2899–2917. doi:10.1016/j.jpdc.2014.06.008.

[18] X. Zeng, et al., IOTSim: A simulator for analysing IoT applications, Journal of Systems Architecture 72 (2017) 93–107. arXiv:1602.06488, doi:10.1016/j.sysarc.2016.06.008.

[19] R. Marler, J. Arora, Survey of multi-objective optimization methods for engineering, Structural and Multidisciplinary Optimization 26 (6) (2004) 369–395. arXiv:3-642-29068-X, doi:10.1007/s00158-003-0368-6.

[20] R. T. Marler, J. S. Arora, The weighted sum method for multi-objective optimization: New insights, Structural and Multidisciplinary Optimization 41 (2010) 853–862. doi:10.1007/s00158-009-0460-7.

[21] K. A. Kaufman, R. S. Michalski, Learning From Inconsistent and Noisy Data: The AQ18 Approach, Symposium A Quarterly Journal In Modern Foreign Literatures (1999) 411–419.

[22] A. Gavras, A. Karila, S. Fdida, M. May, M. Potts, Future Internet Research and Experimentation: The FIRE Initiative, Sigcomm 37 (3) (2007) 89–92. doi:10.1145/1273445.1273460.

[23] K. Deb, et al., Towards estimating nadir objective vector using evolutionary approaches, GECCO 2006 - Genetic and Evolutionary Computation Conference 1 (2006) 643–650.

[24] A. Benítez-Hidalgo, A. J. Nebro, J. García-Nieto, I. Oregi, J. D. Ser], jmetalpy: A python framework for multi-objective optimization with metaheuristics, Swarm and Evolutionary Computation 51 (2019) 100598. doi:https://doi.org/10.1016/j.swevo.2019.100598.
URL http://www.sciencedirect.com/science/article/pii/S2210650219301397