Distributed Task Placement in the Fog: A Positioning Paper

Reinout Eyckerman, Siegfried Mercelis, Johann Marquez-Barja and Peter Hellinckx

Abstract As the Internet of Things (IoT) paradigm becomes omnipresent, so does fog computing, a paradigm aimed at bringing applications closer to the end devices, aiding in lowering stress over the network and improving latency. However, to efficiently place application tasks in the fog, task placement coordination is needed. In this paper, task placement in the fog and corresponding problems are addressed. We look at the fundamental issue of solving Multi-Objective Optimization problems and treat different techniques for distributed coordination. We finish our research with an example to better explain our case and provide some preliminary tests results.

1 Introduction

The Internet of Things (IoT) paradigm is gaining widespread acceptance. A key concept pushing IoT forward is Industry 4.0. One use case within the industrial domain is leveraging IoT to monitor equipment and processes, enabling predictive maintenance and preventing downtime costs. In the automotive sector, another case of IoT usage is smart vehicles, using photodetectors, rain sensors, cameras, and various other connected sensors and actuators. Vinitsky et al. [1] found that only a 10% smart vehicle penetration rate is enough to reduce traffic congestion by 25%, showing that few smart vehicles can already make a difference. To utilize their sensors better, smart vehicles can be connected together to create Vehicle Ad-Hoc Networks (VANETs), allowing vehicles to communicate their information to each other.

However, many sensors mean many devices, which means a massive amount of data putting a serious strain the network. Cisco estimates that by 2021, 850 ZB of

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: {firstname.lastname}@uantwerpen.be

data will be generated yearly, more than triple the amount since 2016 [2]. However, most of this data is short-lived: an estimated 90% of it will be used for processing and will not be stored. This is especially true for IoT devices: the sensor data mainly needs to be processed and acted upon. Fog computing can play a large role in resolving load problems like this. It aims to enable pervasive access to a shared set of computing resources for distributed and latency-aware applications, as defined by Iorga et al. [3]. The placement of the application tasks, will happen in the fog shown in Fig. 1, an intermediate network layer, allowing for a reduced network load and latency. However, fog networks are often highly dynamic. An example here are the previously mentioned VANETs, where vehicles continuously (dis)connect due to differences in speed, or by passing by roadside access points.

Highly dynamic networks demand efficient fog placements, so as not to lose application efficiency when nodes move/disappear. This requires a task placement coordination technique. Applications must first be divided into separate tasks, which can be distributed over the network. The technique keeps in mind the available hardware and network resources along with the dynamic network aspect, and distributes these tasks over the network. The potential overhead and communication cost 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. This context consists of Key Performance Indicators (KPI), which represent device-specific weights inside the network. The KPI 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 will enable application latency reductions, will increased application efficiency, and will also enable automated placement, so that is no longer the concern of the application developer. The research presented in this paper focuses on fog computing, and the distribution of tasks to enable this computing paradigm.

2 State of the Art

The task placement problem spans several areas. Below we present a summary of most relevant research.

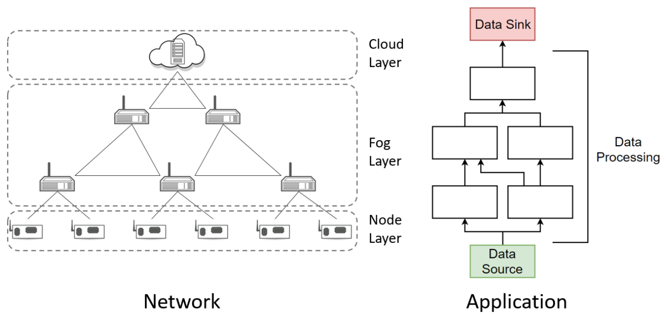

Fig. 1: Example of a fog network & an application graph.

Hardware & Software Models: Xia et al. [4] modeled basic device metrics such as CPU, RAM and disk capacity for a task placement technique on the fog. Huybrechts et al. [5] looked into the application model, focusing on Worst-Case Execution Time (WCET) analysis using a hybrid approach. To achieve their goal, the COBRA and TACLeBench tool suites were used [6]. Using techniques such as these, the application metrics were more efficiently calculated and modeled.

Constraints: Plenty of constraints are present when working with fog devices. Xia et al. [4] considered the resource constraint where the device has a set of available resources, whose usage cannot be exceeded. Similarly, Sharma et al. [7] considered the software heterogeneity when placing tasks on Google compute clusters, with for example tasks requiring a minimum Kernel version to properly run. They then measured the performance impact of such constraints on the task scheduler.

Distributed Software Frameworks: Distributed software development provides more challenges than monolithic software, such as message passing and state maintenance. To aid with this challenge, Vanneste et al. [8] developed the Distributed Uniform STreaming (DUST) framework. The framework provides base building blocks for event streaming in a distributed fashion. This allows users to build parts of applications on these blocks, creating a connected distributed application, which can, in turn, be distributed over the network.

Context: The devices on which the tasks run, employ a context. This concept of context-awareness has been around for some time. One popular definition was coined by Dey & Abowd, whose definition included user, application, location, and device awareness [9]. Although all these contexts influence task placement, only device awareness influences it directly. Location and user awareness can, for example, have an influence on the application, which can, in turn, be simplified into a change in the software requirements. If, for example, it gets dark outside, regular cameras will not be of much use on smart vehicles and can thus be disabled, lowering the amount of data the task has to move, thus changing the task requirements.

Fog Challenges: Many challenges appear when attempting task placement on fog networks, as presented by Wen et al [10]. They described several challenges, such as the IoT device heterogeneity, security, network latency, dynamicity, and fault tolerance, which correspond with problems in the proposed research, such as the device heterogeneity. Others have already attempted similar approaches, such as Brogi & Forti, researching into placing tasks across fog devices regarding the Quality of Service (QoS), and proving NP-hardness [11]. Guaranteeing the QoS was interpreted as placing the tasks without overloading the physical data links between devices and without overloading the machines themselves. Wang et al. [12] proposed a placement algorithm based on Multi-Access Edge Computing (MEC), a technique where they place micro-clouds closer to the edge. They used Linear Programming methods for mapping tree application graphs onto tree network graphs, and make simplifications to make the problem tractable.

Optimization techniques: There are multiple techniques for solving the task placement problem, one of which is by use of heuristics. Xia et al. [4] looked into application placement with fog devices, using dedicated zones for application deployment areas to ensure placed tasks locality. Their goal was to minimize the weighted average latency when placing his tasks over his devices using several solutions, such as exhaustive search, naive search, and improvements of naive search. Wang et al. [12] proposed an MEC application placement algorithm, a technique where they placed micro-clouds closer to the edge, which is then compared to a greedy and the vineyard algorithm. The research used Linear Programming methods for mapping tree application graphs onto tree network graphs. In my previous research, I compared a brute force placement technique to a Genetic Algorithm (GA) and a Hill-Climb (HC) technique [13]. Plenty of research already exists, but it is noticeable a distributed implementation is lacking. Distributed approaches bring massive added complexity. One method of solving the problem is by using the Contract Net Protocol, as defined by Smith [14]. There, the agents distributed tasks one by one, using a simple bidding mechanism. Another solution would be the A-Team, as defined by Talukdar et al. [15], who defined a set of autonomous agents, called the A-Team or Asynchronous Team, which work together to solve the same goal. This approach was implemented by Barbucha & Je [16], to solve the Vehicle Routing Problem. Barbucha also showed the efficiency of distributed agent optimization [17]. Here they focus on placing teams of agents on a distributed network and showing that an increase of agents improves efficiency up to a certain point, after which efficiency starts decreasing again due to overhead.

Simulation: The research is to be tested using simulation. One versatile simulator is Simgrid, which has been in active development since 2001 [18]. This simulator allows the simulation of a concurrent software stack on top of simulated hardware resources, optionally using 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 such as IoTSim [19], 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 problems not stated in the state of the art. Different kind of placement techniques will need to be compared and reviewed, combining and editing where necessary, to finally end up with a stable technique.

3 Problem Definition

The main objective of this paper is to show the current problems involved when trying to distribute tasks across IoT networks using decentralized optimization algorithms. There are several reasons for not using a centralized approach. A centralized approach might be too slow for time-critical applications (optimization or communication might take too long), in a highly dynamic environment a single point of failure is quite dangerous, and furthermore, centralized monitoring of global resource availability in real-time introduces plenty overhead. We define the metrics used for the application and network and describe which shapes of application and network graphs are supported. After this, we define our context and the Multi-Objective Optimization (MOO) problem. Finally, we describe the features an efficient technique should have, and a use case.

Metrics: Metrics for both the software application and the network hardware must be defined. The selected network metrics can be seen in Table 1, where the metrics of the nodes and links in the network are defined. The application metrics are defined in Table 2. It can be seen that the defined software 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 Network Interface Card (NIC) transfer speed of the device as to the bandwidth of the link. Most other metrics are usually network specific, such as wireless network interference. One exception is the data storage cost. However, streaming applications are assumed, where no data needs to be stored on the device. The network model can be created by using the hardware characteristics. The application model 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
cpu_n	Processing Speed of node *n*
mem_n	Memory Size of node *n*
nic_n	NIC Transfer Speed of node *n*
ec_n	Energy Consumption of node *n*
bw_l	Bandwidth (kbps) of link *l*
lat_l	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
$wcet_a$	Worst-Case Execution Time of task *a*
$wcmc_a$	Worst-Case Memory Consumption of task *a*
$wcec_a$	Worst-Case Energy Consumption of task *a*
$wcnl_c$	Worst-Case Network Load of data link *c*
$mlat_c$	Maximum Allowed Latency of data link *c*

Network & Application Shape: Both the network shape and the application shape will be modeled as graphs. Previous research often used tree graphs for the network, and directed linear [13] or tree graph structures for the application [12]. Constraining the graphs allows for constraint modeling into the optimization techniques. Mesh-like networks are underrepresented due to them greatly enlarging the search space but are necessary to exploit the full benefits of the network, such as redundancy. Similarly, directed mesh application graphs are required, in order to support any kind of application. An example of a network and application graph is shown in Fig. 1. As observed, the bottom application node represents a data source or sensor, the top node a data sink or actuator.

Context: There is a device-specific context assigned to each device. This context is a set of KPI, each connected to their respective device parameter. These contexts allow the device to tune itself in regards to task placement. If for example, the device has a low battery, the energy KPI goes up, stopping energy-hungry tasks to be assigned to the device.

Multi-Objective Optimization: Determining the optimal task placement on the network results in a Multi-Objective Optimization (MOO) problem. It is an optimization problem where multiple, potentially conflicting, criteria are to be optimized simultaneously using multiple objective functions, e.g. optimizing toward energy efficiency and maximizing machine utilization. Solving this tends to result in a set of optimal solutions, where no criteria can be improved without degrading other criteria. This set of optimal solutions is called the Pareto front. In this scenario, each optimization criterium (e.g. energy efficiency, latency) can be modeled into an objective function, which results in a MOO problem. Marler & Arora [20] provide a survey paper about available techniques. There are three main techniques for working with MOO problems.

- A priori: preferences are modeled in advance in order to achieve a subset of Pareto-optimal solutions.
- A posteriori: the entire Pareto-optimal set is determined, after which one selects a solution based on preferences.
- Interactive method: multiple method iterations are done, aiding the user with a selection of preferences.

The coordination technique should be able to work autonomously, placing applications without human interaction or feedback. In order to be able to work autonomously, the coordination technique is required to know the preferences of placements, such as valuing energy efficient placements over low latency placements. This is application and network dependent, and thus is the system administrator's task to determine this. However, this requires that the 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. In this regard, only the *a priori* method can be used for it is the only technique that can provide a single solution, because it models the MOO problem as a single-objective problem, either by defining weights, preferences or other techniques. The approach does often have problems in finding Pareto-optima, depending on how well the function is constructed. Two *a priori* methods are defined below:

Weighted Sum: A simple and widely used approach is the weighted sum approach, where the set of objectives is scalarized into a single objective function by multiplying each objective function with a user-defined weight. Marler & Arora [21] list up several problems with this approach, one of them being objective function normalization. This simplifies 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.

Lexicographic Method: When using the lexicographic method, the objective functions are sorted and solved in order of importance. Multiple techniques exist in aiding prioritization of objective functions, such as presented by Kaufman & Michalski [22]. They present a technique where multiple parameters are defined per objective function which corresponds to the results they present in the set of Pareto solutions.

3.1 Optimization Technique Features

We will now define the most prominent features which should be kept in mind during the research process is in order. These features contain distributivity, speed, resource consumption, memory, scalability, and optimality. The features have an impact on the technique's efficiency for the application placement problem and can be used as measurements.

Distribution: Important is that the technique can run in a distributed manner. Little communication overhead is required as to not overload the network links. It should run competitively on smaller devices so that they can add to finding the solution. Stronger devices should not need to wait on solutions or skip the smaller devices. It should be fault tolerant so that there still can be found a solution when a device misbehaves/fails.

Speed: Another feature to keep in mind is the speed of the used technique. Working with a highly dynamic network, both on context as on device level, solutions should be found quickly. Otherwise, the distributed tasks might perform badly or devices get overloaded with tasks.

Resource Consumption: The resource consumption is of grave importance. If the technique consumes too much memory or processing power while running, the device might not be able to process its other tasks productively. In a highly dynamic network, this would result in devices being unreliable.

Scalability: This feature asks that the technique is scalable on a heterogeneous network. A major issue here is that the Internet of Things (IoT) network is heterogeneous by nature, where devices and links have different resources available. Due to energy depletion, link failure or any other kind of problem, the devices can disconnect, resulting in a continuously changing network.

Solution Memory: Solution memory is 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. 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.

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). It should be kept in mind that if global optimization techniques are not feasible, they should be discarded as soon as possible.

3.2 Techniques

There are multiple techniques that can be used, as stated in the state of the art. Due to the design of the application and network graphs, there is a very large search space with worst-case $O(N^A)$ run time, meaning exact search techniques are infeasible. Heuristics can bypass this problem. However, finding and composing the best heuristic is a major part of this 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 heuristics for pre-optimization, improving search efficiency. Another selection to be made is the multi-agent technique, such as A-Team, combining it with heuristics.

3.3 Simulation

The Simgrid simulator platform shows to be interesting for testing the approaches. However, simulation is limited in its results. To test the distribution techniques, a set of VMs could be modeled into the network to be simulated. These VMs can model the resources available by the devices, and for example, the Linux *netem* command can simulate network links between the VMs so to represent the real connection links. Using this VM technique, the effect of both the distribution technique as that of the application distributed can be observed. Improved realism and scalability can be achieved using a test-bed, using, for example, the Fed4Fire test-bed [23]. Use of a test-bed, however, is time-consuming, and should thus be well-prepared by simulation before actual deployment.

3.4 Use Case

As use case we take the use case we proposed in our previous work [13]. We adapt the network and application graph from the original scenario, in order to provide mesh networks and mesh graphs. The network is shown in Fig. 2. It describes an industrial plant, where there is a camera monitoring the area for any problems occurring, such as fire, and using facial recognition to look for trespassers. In case of any problems, alerts are sent to every other device in the visualized network. 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 Test

We have provided a small test scenario as some preliminary results. First, we describe our test scenario, after which we explain our achieved results. In this scenario, we test centralized coordination techniques and compare them. This differs from distributed techniques since a single node is responsible for the placement. A simplified scenario is provided, which differentiates from future research since cur-

rently, placement calculation happens centralized.
The use case is the previously defined use case, shown in Fig. 2. The network is a static network, purely for testing placement technique. Four different contexts are used, where the cloud, edge devices, actuators, and monitoring devices have a different context. The green node represents the video monitor as a data source, and the red nodes are the actuators as data sinks. The blue node is a task forced to run at the cloud level, for data integrity checking. All the colored nodes are locked nodes, and thus cannot be moved. The Key Performance Indicators (KPI) are defined with a preference to keep the tasks as close to the node level as possible. The objective function we use is the following, based on [13].

$$C = \sum_{i=0}^{\#Components} \sum_{j=0}^{\#KPI} w_{ij} C_{ij} \tag{1}$$

Here we try to minimize the placement cost of component i onto the device with 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.

Branch & Bound: A branch & bound with fail-early constraint checking is used to provide a baseline. A constraint is, for example, a device is allocated more tasks than its resources can handle. This algorithm is guaranteed to return the best placement, by iteratively checking all possible placements. However, since it has to check all possible placements, it has a worst-case run time of $O(N^A)$, making it is infeasible to use this technique in practice.

Hill Climb: The Hill-Climb (HC) heuristic is used as a local optimization algorithm. It is based on a multiple restart steepest descent HC, implemented with 10 restarts. It attempts to improve total cost by greedily moving tasks to their neighboring device where possible.

Genetic Algorithm: A GA is tested as well. 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. The single-point crossover function splits the placed task set in two and shares the

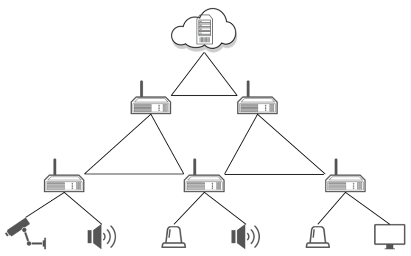

Fig. 2: Use Case Network Graph

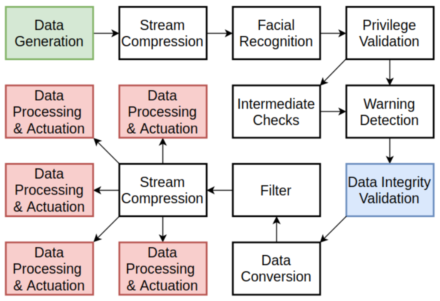
Data Generation
Stream Compression
Facial Recognition
Privilege Validation
Data Processing & Actuation
Data Processing & Actuation
Intermediate Checks
Warning Detection
Data Processing & Actuation
Stream Compression
Filter
Data Integrity Validation
Data Processing & Actuation
Data Processing & Actuation
Data Conversion

Fig. 3: Use Case Application Graph

first part. Parents for this function are selected using roulette-wheel selection. The mutator function changes a randomly selected task to a randomly selected device.

4.1 Test Setup & Results

The used hardware is a server with 2x Intel Xeon E5-2420 v2 CPU and 32GB RDIMM memory. Next to the use case, random networks were developed with an

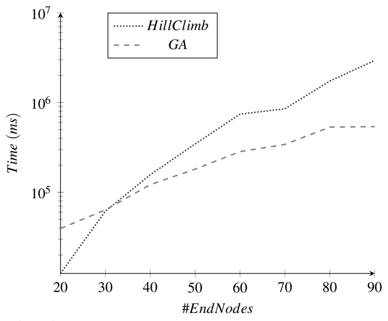

Fig. 4: Algorithm Running Time with twice as much free tasks as nodes

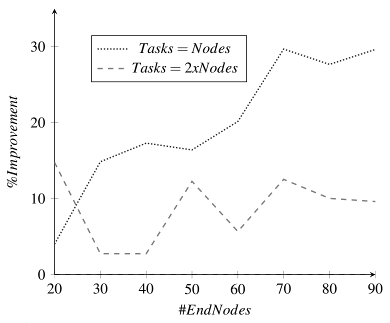

Fig. 5: Solution improvement of GA compared to HC

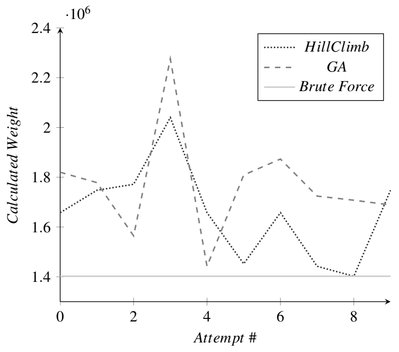

Fig. 6: Comparison of calculated weight on use case

increasing amount of nodes. For each of these networks, we generated two applications, which always included all sensors and actuators as either start or end nodes, and forced at least one task at the cloud. The first of these two applications has the same amount of free 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.

The first application showed that run-time for HC and the GA where comparable, and as thus are not shown for brevity. In Fig. 4 it is shown that the HC runtime increases considerably faster with the network size, compared to the GA who seems to be following a logarithmic trend. This is to be expected since the operations of the GA are more costly but are more consistent than those of the HC, which only moves a single task each iteration.

The GA is able to search for an optimum utilizing the complete set of tasks and the complete network at the same time, allowing it to handle much larger search spaces with much less time. However, the solution quality is another important metric, where Fig. 5 shows the solution improvement in % of the GA compared to the HC. This improvement on small application graphs is due to the randomization of the GA, giving it more room to play than the HC. This improvement decreases the more tasks there are, allowing HC to continuously improve. A potential improvement is the Simulated Annealing algorithm, allowing the HC to exit local optima, requiring fewer restarts, improving both runtime and found optima.

In the search space with the least amount of tasks, the GA performs considerably better, but the unstable lines show both algorithms have trouble finding their solutions in the larger search space, being greatly influenced by the randomness. Finally, looking at the use case scenario in Fig. 6, we see that on a small scale HC manages to find the optimum, while the GA hovers above it. This is due to the HC taking individual steps, allowing the last task to shift into position, whereas the GA uses its population-based technique to get as close as possible. Due to the small network, the HC performs considerably better.

5 Conclusion & Future Work

Fog computing will help in relieving network stress, but this needs a coordinator function to organize the fog processes. 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 and more consistent results. 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, and A. Bayen, “Lagrangian Control through Deep-RL : Applications to Bottleneck Decongestion,” *IEEE Intelligent Transportation Systems Conference*, pp. 759–765, 2018.
2. Cisco, “Cisco Global Cloud Index : Forecast and Methodology 2014-2019 (white paper),” *Cisco*, pp. 2016–2021, 2016.
3. M. Iorga *et al.*, “Fog Computing Conceptual Model,” NIST, Gaithersburg, MD, Tech. Rep., mar 2018.
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)*, pp. 153–163, 2018.
5. T. Huybrechts, S. Mercelis, and P. Hellinckx, “A New Hybrid Approach on WCET Analysis for Real-Time Systems Using Machine Learning,” no. 5, pp. 1–5, 2018.
6. IDLab, Imec, and University of Antwerp, “COBRA Framework.” [Online]. Available: http://cobra.idlab.uantwerpen.be/
7. B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and 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.
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*, pp. 426–436, 2019.
9. G. D. Abowd *et al.*, “Towards a better understanding of context and context-awareness,” *Lecture Notes in Computer Science*, vol. 1707, pp. 304–307, 1999.
10. Z. Wen *et al.*, “Fog orchestration for internet of things services,” *IEEE Internet Computing*, vol. 21, no. 2, pp. 16–24, 2017.
11. A. Brogi and S. Forti, “QoS-aware deployment of IoT applications through the fog,” *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1–8, oct 2017.
12. S. Wang, M. Zafer, and K. K. Leung, “Online Placement of Multi-Component Applications in Edge Computing Environments,” *IEEE Access*, vol. 5, pp. 2514–2533, 2017.
13. R. Eyckerman, M. Sharif, S. Mercelis, and P. Hellinckx, “Context-Aware Distribution In Constrained IoT Environments,” 2019, pp. 437–446.
14. R. G. Smith, “Communication and Control in a Distributed Problem Solver,” *IEEE Transactions on Computers*, vol. C, no. 12, pp. 1104–1113, 1980.
15. S. Talukdar, L. Baerentzen, A. Gove, and P. De Souza, “Asynchronous Teams: Cooperation Schemes for Autonomous Agents,” *Journal of Heuristics*, vol. 4, no. 4, pp. 295–321, 1998.
16. D. Barbucha and P. Je, “An Agent-Based Approach to Vehicle Routing Problem,” *International Journal of Applied Mathematics and Computer Science 4.1*, vol. 1, pp. 36–41, 2007.
17. D. Barbucha, “Agent-Based Optimization,” in *Agent-Based Optimization*, 2013, vol. 456, pp. 55–75.
18. H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, “Versatile, scalable, and accurate simulation of distributed applications and platforms,” *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, 2014.
19. X. Zeng *et al.*, “IOTSim: A simulator for analysing IoT applications,” *Journal of Systems Architecture*, vol. 72, pp. 93–107, 2017.
20. R. Marler and J. Arora, “Survey of multi-objective optimization methods for engineering,” *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, apr 2004.
21. R. T. Marler and J. S. Arora, “The weighted sum method for multi-objective optimization: New insights,” *Structural and Multidisciplinary Optimization*, vol. 41, pp. 853–862, 2010.
22. K. A. Kaufman and R. S. Michalski, “Learning From Inconsistent and Noisy Data: The AQ18 Approach,” *Symposium A Quarterly Journal In Modern Foreign Literatures*, pp. 411–419, 1999.
23. A. Gavras, A. Karila, S. Fdida, M. May, and M. Potts, “Future Internet Research and Experimentation: The FIRE Initiative,” *Sigcomm*, vol. 37, no. 3, pp. 89–92, 2007.