Metis: Optimal Task Allocation Framework for the Edge/Hub/Cloud Paradigm

Andreas Kouloumpris, Theocharis Theocharides and Maria K. Michael
Department of Electrical and Computer Engineering,
KIOS Research and Innovation Center of Excellence,
University of Cyprus
{ kouloumpris.andreas, ttheocharides, mmichael}@ucy.ac.cy

ABSTRACT
Increased demands for real-time decision support and data analytics facilitate the need of performing significant computing away from the cloud and onto the IoT devices. In this paper, we propose Metis, a mathematical programming based framework, able to deliver an optimal task allocation when targeting a specific performance metric. Metis is currently suitable for systems which consist of an edge node, an intermediate node and the cloud infrastructure, but can be expanded to multi-Edge/Hub systems. Evaluation results using a real-life use-case scenario demonstrate that Metis provides the optimal task allocation by minimizing the overall latency of the system while taking into consideration the application’s specific requirements and resource constraints of each computational unit.

CCS CONCEPTS
• Theory of computation → Mathematical optimization; Continuous optimization; • Computer systems organization → Embedded and cyber-physical systems; Real-time systems.

KEYWORDS
Optimal Task Allocation, Mathematical Programming, Real-life use-case Scenario, Edge-Hub-Cloud Paradigm

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Andreas Kouloumpris, Theocharis Theocharides and Maria K. Michael. 2019. Metis: Optimal Task Allocation Framework for the Edge/Hub/Cloud Paradigm. In INTERNATIONAL CONFERENCE ON OMNI-LAYER INTELLIGENT SYSTEMS (COINS), May 5–7, 2019, Crete, Greece. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3312614.3312643

1 INTRODUCTION
Traditionally, power hungry and heavy computing tasks have been executed on powerful servers. As the number of connected and portable devices grows, and their application spectrum rapidly expands, the centralized computing model has been altered. Traditional servers have been replaced by cloud infrastructure, and intermediate nodes, which perform most of the processing/analytics. In the past few years, the number of IoT (Internet of Things) devices increased exponentially and is expected to surpass the 50 billion by 2020 [4]. The overall infrastructure is now integrating these devices, thus yielding the edge/hub/cloud paradigm. This paradigm integrates the edge devices which are not just sensing devices, instead, they also have computing capabilities. Moreover, as emerging applications require real-time decision support and data analytics, there is an increased need for performing significant computing on these edge devices. This paradigm is relatively new but it has significant challenges that must be overcome. One of these challenges is the task allocation, in terms of which tasks should be executed at the edge, hub or cloud.

A number of studies [2, 7, 11, 14], show that by shifting tasks from a central cloud data center towards the edge nodes enables real-time data processing, and results in a network with reduced traffic. However, as tasks move towards the edge of the network, the nodes become resource constrained, both in terms of computation capability and memory capacity. Hence, although it makes sense to allocate all application’s tasks at the edge node to minimize latency, due to the uniqueness of every application, the allocation must be decided based on these unique characteristics. Performance, and more specifically latency, is essential for time-critical applications, such as critical infrastructure monitoring and search-and-rescue applications, in which even a fractional delay in processing can trigger a disastrous event. Therefore, one might need to further analyze the allocation of each task between the different computational units, to maximize the performance.

To allocate an application’s tasks between the different computational units of the paradigm model, we need to consider several constraints, both in the application but also in the infrastructure itself. However, an optimal task allocation with tasks that require energy, memory and execution time, remains a challenge, primarily because the edge/hub/cloud paradigm has significant constraints that must be taken into account.

We, therefore, propose Metis, a framework which uses a Mathematical Programming (MP) approach to optimally partition an application’s tasks within the paradigm, to achieve maximum performance, in terms of latency. In a nutshell, Metis yields the optimal task allocation such that the overall latency of the system is minimized under energy and resource constraints for each one of the computational units. To do so, Metis takes into account the execution time, the energy consumption, the memory footprint and the storage requirements of the application’s tasks. Furthermore, it
takes into consideration the communication latency and the communic-
ination energy, which are computed based on the load generated
from each of the application’s tasks as well as the bandwidth of the
communication channel. Metis is suitable for a range of applica-
tions that involve remote decision and support, machine learning
inference applications, health-care applications, e.t.c., where an
optimization within the edge/hub/cloud paradigm is a necessity.

The remaining sections of the paper are organized as follows. In
section 2, preliminaries and existing related work are discussed.
In section 3, a detailed description of our framework and a problem
formulation are presented. In section 4, the results and the evalua-
tion of the implementation based on a real-life use-case are given,
followed by concluding remarks in section 5.

2 BACKGROUND AND RELATED WORK

Notwithstanding that a large number of studies approached the
subject of task allocation and investigated the idea of computational
offloading for several years, they dealt mostly with the edge-cloud,
fog-cloud computing paradigm. Recently, Ghosh et al. [5], formu-
lated an optimization problem for energy-aware placement of Com-
plex Event Processing queries across a collection of edge and cloud
resources, with the goal of minimizing the end-to-end latency. They
proposed a brute force approach and a genetic algorithm pattern
to solve the problem. Loke in [10] examined the chance of hand-
ning off jobs to other devices through communication interfaces.
In addition, in [15] Wang et al. presented a cross-end analytic engine
architecture for wearable computing systems. They proposed three
design rules to optimize energy efficiency for a single functional
cell and they developed an automatic generator to find an optimal
partitioning for cross-end architecture.

In [12] Ouahouah et al. suggested a computation offloading
scheme amongst unmanned aerial vehicles (UAVs) that handle IoT
tasks. Their purpose was to enhance the UAVs’ lifetimes and re-
duce their response times. In [8], Kovachev et al. introduced an
adaptive computation offloading middleware for mobile devices
to offload their computation tasks to a cloud server. Their results
showed that the local execution time can be substantially reduced
by computation offloading. In [3] Dinh et al., introduced an op-
imization framework of offloading from a single mobile device
(MD) to multiple edge devices. Here, they minimized the total cost
which included both the MD’s energy consumption and total tasks’
execution latency. Additionally, in [13], Ryden et al. reduced the
execution time by aiming to pick the best-performing server nodes
for computation. To minimize the overall system’s execution time,
an optimal task allocation is required.

Most of the studies focusing on the allocation of computing
resources, consider a mobile edge computing architecture where its
computing nodes have the same computing capability. Moreover,
regarding the allocation of tasks and computing resources as well as
the computation offloading, recent researches typically ignore the
fact that the offloaded data has to be transmitted over the network.
Therefore, it is necessary to take into account the communication
resources when dealing with applications where data offloading is
a part of them.

The edge/hub/cloud paradigm is newly introduced, therefore,
no standard architecture is available regarding the management
and allocation of resources and tasks across the paradigm. Our
work proposes a mathematical programming (MP) based scheme
for achieving an optimal partitioning of an application across the
edge/hub/cloud paradigm model by minimizing the overall latency
of the system, taking into account the latency and energy consump-
tion of both nodes and communication links, as well as the memory
needed for the tasks to be executed.

3 METIS FRAMEWORK

3.1 Extended Task Flow Graph

We propose an extended task flow graph as the input for our frame-
work. The extended task flow graph (sample is shown in Figure 1) consists of three components:

1) The first component is the group of candidate tasks. For each
task specified in the application’s description, there is a group of
candidate tasks (i.e. a set of candidate tasks \( S_c \), where \( n \)
is the number of computational units), which indicates the possible
computational units where the task can be allocated. Hence, the
extended task flow graph contains one node per candidate task.
Each candidate task has its own unique characteristics in terms of
computational and energy consumption and eventually, only one
candidate task from the group will be selected for execution. The al-
location of each candidate task specified in the application’s descrip-
tion is based on the application’s constraints and the constraints
imposed by each one of the computational units.

2) The second component involves the connectivity between the
tasks, that is, the edges of the graph. Each edge indicates the
dependency between two tasks, as well as the communication cost
(computation latency and communication energy), which is cal-
culated based on the data generated by each candidate task and the
bandwidth of the communication channel. In our case we assume
wireless communication between the nodes; however, our model
is generic and it can easily be applied to wired communication
protocols as well.

3) The third and final component is an extra communication
cost (communication latency and energy) in case there is no direct
communication between a pair of computational units. In this case,
an intermediate node will act as a communication bridge between
the two candidate tasks.

3.2 Properties of Candidate Tasks

Each vertex of the extended task flow graph denotes a candidate
task \( T \). Each candidate task \( T \) has the following six attributes: (1)
the computation latency which is the time required for the task
to complete, (2) the energy consumption which is the energy re-
quired for the task to complete, (3) the memory footprint, that is,
the amount of RAM memory needed for the task to run, (4) the
secondary memory requirements which is the disk storage occu-
bied by the task, (5) the generated data by task, and (6) the set of
predecessors of the task.

3.3 Energy Model

We use a similar energy model as the one used in [15], where
the system energy consumption \( E \) can be split into two main parts: the
energy needed to execute the allocated tasks, \( E_T \), and the energy
needed for transmitting data between the computational units,
hence the communication energy, $E_c$, as shown below:

$$E = E_x + E_c$$  \hspace{1cm} (1)

The following equation illustrates the energy model for the execution of the tasks:

$$E_x = \sum_{i=1}^{s} P_t \times t_i,$$  \hspace{1cm} (2)

where $P_t$ and $t_i$ are the power consumption and the execution time of the candidate task $T_i$, respectively. The constant $s$ indicates the number of the allocated candidate tasks, that is, the number of the application's original tasks.

We also model the energy consumption of transmitting and receiving data into our wireless communication energy consumption:

$$E_c = B_r \times A_r + B_s \times A_s,$$  \hspace{1cm} (3)

where $B_r$ is the total number of bits for transmitting and $B_s$ is the total number of bits for receiving. The $A_r$ and $A_s$ parameters are the average energy model for one-bit data transmission and one-bit data reception, respectively.

### 3.4 Properties of graph edges

Each edge of the extended task flow graph denotes a communication edge between two candidate tasks, e.g., $T_i \rightarrow T_{i+1}$. Its attributes are the communication latency and communication energy. To calculate the communication energy and latency, the bandwidth $B$ of each one of the communication channels must be defined. We also need the output data $D$ that is generated and transmitted from each candidate task. Let $T_i T_j, \forall i, j \in \{1, \ldots, s\}$, where $s$ is the number of the application’s tasks, denote the candidate task $T_i$ that transmits and candidate task $T_j$ that receives the data respectively and $D_T$ the data generated from the candidate task $T_i$. It should be noted that each task $T_i$ encapsulates the information concerning to which computational unit it corresponds. Moreover, let $U_x U_y$ denote the computational unit $U_x$ that transmits the data and computational unit $U_y$ that receives the data. Moreover, let $B_{U_x U_y}, \forall x, y \in \{1, \ldots, n\}$, where $n$ is the number of computational units, denote the bandwidth of the channel $U_{x} \rightarrow U_{y}$.

The communication latency ($C_{L_{T_i T_j}}$) is defined as follows:

$$C_{L_{T_i T_j}} = D_T / B_{U_x U_y}$$  \hspace{1cm} (4)

This can be re-written as follows:

$$C_{L_{T_i T_j}} = D_T / B_{U_x U_y}$$  \hspace{1cm} (5)

Equation (5) gives the communication latency required for task $T_i$ allocated to Computational Unit $U_x$ to send data to Computational Unit $U_y$.

The communication energy needed for the transmission ($C_{E_{T_i T_j}}$) and reception ($C_{E_{T_i T_j}}$) of the data are defined as follows:

$$C_{E_{T_i T_j}} = D_T \times A_t \times C_{L_{T_i T_j}}$$  \hspace{1cm} (6)

$$C_{E_{T_i T_j}} = D_T \times A_r \times C_{L_{T_i T_j}}$$  \hspace{1cm} (7)

As communication latency, the communication energy can be re-written as follows:

$$C_{E_{T_i T_j}} = D_T \times A_t \times C_{L_{T_i T_j}}$$  \hspace{1cm} (8)

$$C_{E_{T_i T_j}} = D_T \times A_r \times C_{L_{T_i T_j}}$$  \hspace{1cm} (9)

### 3.5 Problem Formulation

#### 3.5.1 Variables in the MP formulation

To formulate the problem, let $i$ denote a candidate task $T_i$ and $j, k$ denote a computational unit $U_j$ and $U_k$, respectively. Also, let $E_{ij}, L_{ij}, M_{ij}$ and $S_{ij}$ be the energy, latency, memory footprint and secondary memory requirements of the $T_i$ candidate task allocated to be executed on the $U_j$ computational unit, respectively, where $E_{ij}, L_{ij}, M_{ij}$ and $S_{ij} \in \mathbb{R}_0^*$. Moreover, let $C_{L_{ijk}}$ and $C_{E_{ijk}}$ be the communication latency and communication energy needed for $T_i$, which is allocated to be executed on $U_j$ and sends a specific amount of data to $U_k$, where $C_{L_{ijk}}$ and $C_{E_{ijk}} \in \mathbb{R}_0^*$. Given the extended task flow graph $G$, $E_{ij}, L_{ij}, M_{ij}, S_{ij}, C_{L_{ijk}}, C_{E_{ijk}}$ and the energy budgets $E_j \in \mathbb{R}_0^*$ together with the memory budgets $M_j \in \mathbb{R}_0^*$ and secondary memory budgets $S_j \in \mathbb{R}_0^*$, we want to find the optimal task allocation such that the overall latency $L$ is minimized. The model is problem using MP, with the following variables:

- A binary variable $x_{ij}$ as $x_{ij} = 1$ if task $i$ is allocated to be executed on unit $j$, and 0 otherwise.
- A binary variable $x_{ijk}$ as $x_{ijk} = 1$ if the outgoing link $j \rightarrow k$ of task $i$ is activated, and 0 otherwise.
- A variable $p$, which indicates the difference between the incoming and the outgoing links on a node. In our formulation we define a super-source node which communicates directly with tasks with no predecessors and a super-sink node which is reached directly with all nodes with no immediate successors. This is not shown in the extended task flow graph (fig. 1).

$$p = \begin{cases} 0, & \forall i, j, k \\ -1, & \text{source} \\ 1, & \text{sink} \end{cases}$$  \hspace{1cm} (10)

#### 3.5.2 Objective Function

The Objective Function of the MP is to find a solution that minimizes the overall latency of the system. Let $L$ be the set of the candidate tasks of the extended task flow graph and $J$ be the set of the computational units.

$$\min \left( \sum_{i \in L} \sum_{j \in J} L_{ij} x_{ij} + \sum_{i \in L} \sum_{j \in J} \sum_{k \in J} C_{L_{ijk}} x_{ijk} \right)$$  \hspace{1cm} (11)

subject to

$$x_{ij} \in \{0, 1\}, \forall i, j$$  \hspace{1cm} (12)

$$\sum_{j \in J} x_{ij} = 1, \forall i$$  \hspace{1cm} (13)

$$x_{ijk} \in \{0, 1\}, \forall i, j, k$$  \hspace{1cm} (14)

$$\sum_{j \in J} \sum_{k \in J} x_{ijk} = 1, \forall i$$  \hspace{1cm} (15)

$$\sum_{k \in J \setminus \{j, k\}} x_{ijk} = x_{ij} \forall i, j$$  \hspace{1cm} (16)

$$\sum_{k \in J \setminus \{i, j\}} x_{ijk} = \sum_{k \in J \setminus \{i, j\}} x_{ij} \forall i, j \cup \{\text{source, sink}\}$$  \hspace{1cm} (17)

$$\sum_{i \in L} M_{ij} x_{ij} \leq M_j, \forall j$$  \hspace{1cm} (18)
Equations (12) and (13) indicate that each task \( i \) is assigned only to one of the computational units \( j \). Equations (14) and (15) indicate that only one of the outgoing links \( j \rightarrow k \) of task \( i \) will be activated each time. Equation (16) correlates the two decision variables. Equation (17) indicates the flow balancing constraint, that is, if there is an ingoing link to a task \( i \) then it must also have an outgoing link. In this way, the path between the nodes is guaranteed to be connected. Equations (18) to (20) indicate the resource constraints. For the main and secondary memory usage as well as the energy consumption we use a simple summation of the memory, physical storage, and computation and communication energy needed by each task \( i \), respectively. At any given time, the total main and secondary memory usage along with the total energy consumption should not exceed the specified memory and energy budgets. If a constraint is violated, Metis is able to identify this and report that there is no solution due to a constraint violation. It should be noted that there is no formulated constraint in the event that a task cannot be executed on a specific computational unit, but this is encoded in the extended task flow graph instead.

4 EXPERIMENTAL EVALUATION AND RESULTS

4.1 Case Study

To evaluate Metis, we use a real-life problem that involves an autonomous UAV-based power line inspection. The UAV first detects a pylon and then follows the power lines in an attempt to detect transmission line anomalies, such as vegetation, faulty insulators, etc. The problem is very relevant to our targeted applications as it involves a dynamic decision and support system at the UAV, which is also the edge device. The application consists of two main parts as shown in Figure 2. The first part is the power transmission lines detection process and the second major part is the pole and pylon detection process. The power transmission lines detection process takes as input an image captured from the UAV’s camera and then uses a Hough transform based approach to detect the power transmission lines. The main component of the pylon and pole detection process is the detector based on a convolutional neural network (CNN) [9]. Overall, the process involves several tasks which can run either on the UAV, on a hub node such as a ground station or on a remote server (i.e. cloud).

Metis receives as an input the extended task flow graph to generate the optimal solution. Based on the application’s constraints, the groups of candidate tasks are constructed to create the extended task flow graph. Next, using profiling tools and a digital energy monitoring device we computed the various metric values for each candidate task such as the latency, the energy consumption, the memory footprint, the storage requirements, as well as, the communication latency and communication energy, depending on the targeted computation and communication platform. More details are provided in section 4.2. Using these values, we generated the extended task flow graph. It is worth mentioning that in our use case, there is no direct communication between the edge node and the cloud unit. Therefore, this connection can be established only through an intermediate node, in our case the hub node (ground station).

Figure 2 shows the application flowchart, which can also correspond to the original task flow graph, and Figure 1 shows the extended task flow graph used by Metis. Observe that tasks \( T_1 \) and \( T_{15} \) in Figure 2 can only be executed on the edge and hub devices, respectively. This is reflected in the extended task flow graph in Figure 1, where there is a single node \( T_{1e} \) for \( T_1 \) and a single node \( T_{15h} \) for \( T_{15} \).

It should be noted that any of the UAV’s flight control computation has been excluded from our framework as it involves the UAV’s flight capabilities and is not part of the application.

4.2 Experimental Setup

For our setup, we used four different computational devices, the Odroid XU4 and the Raspberry Pi 3 (RPi3) as the possible edge...
devices that can be attached as UAV’s payload to perform the computation, and the Mi Notebook Pro and the Samsung Tab S2 as the possible hub devices. The chosen devices have been selected because they belong to a family of devices that are typically deployed in these use-case scenarios and they offer different computation and communication characteristics, enabling a qualitative and quantitative evaluation of our framework. For the cloud unit, we used a server equipped with an Intel Xeon E5-2670v2. To evaluate our system we used four different configurations using the different computational devices, two for the edge node and two for the hub device, while the cloud unit remained the same for all configurations. Table 1 shows in detail the various configurations we used to test our framework as well as the parameters and constraints for the edge and hub devices and the cloud unit.

Furthermore, we ran our framework twice for each configuration, to reflect for any bandwidth variations, depending on where the application is deployed, e.g., rural, urban, and the distance between the nodes [1]. During the first run the bandwidths of the communication channels Hub → Cloud and Cloud → Hub are set at 2.0x compared to the second run. This is shown in Table 2. To solve the formulated problem and to find the optimal solution, we run the Gurobi Optimizer 8.1 [6] on an Intel® Core i5-6500™ CPU @ 3.20 GHz (Memory: 7.7 GiB of RAM). The solution for the given extended task flow graph was extracted in the order of milliseconds (130ms).

### 4.3 Results and Discussion

Figure 3 and Figure 4 report the overall latency of the system for each of the four configurations per run. More specifically, they illustrate the computation latency for each computational device edge, hub, and cloud as well as the communication latency of each communication channel edge → hub, hub → edge, hub → cloud, and cloud → hub, that occurred during the execution of the algorithm for some specific task allocation. The computation (communication) latency components are given in solid (patterned) color. It should be noted that for each configuration (A, B, C, D) we consider four different scenarios of task allocation, that is, the three extreme cases where, if possible, all tasks are allocated to a single computational unit, and the allocation of Metis. More specifically, in each of the three extreme cases, all tasks, if possible, were allocated only at the edge, hub or cloud, respectively. Only tasks $T_1$ and $T_{15}$ have a fixed allocation, as explained in Section 4.1.

For each configuration per run, the results show that Metis returns the optimal task allocation, that is, minimizing the overall latency of the system. In Run 1 (fig. 3), the second-best performance in terms of latency, was the case in which all tasks were allocated at the hub device, while the worst performance was provided by the case in which all tasks were allocated at the edge node. This was evident in all four configurations. In Run 2 (fig. 4), the second-best performance in terms of latency, for all configurations, was again the case in which all tasks were allocated at the hub device. The worst performance, though, was observed in the scenario where all tasks were allocated at the edge node. This was due to the fact that the Odroid XU4 is faster than the RPi3 and the overall computation latency cannot overcome the communication latency to the cloud.
5 CONCLUSION & FUTURE WORK

In this work, we proposed a novel framework, Metis, to optimally allocate an application’s tasks into the different computational units of the edge/hub/cloud paradigm. Specifically, we introduced the extended task flow graph, a more suitable way to represent an application’s tasks for this kind of problems. Furthermore, we presented an MP formulation of the task allocation problem and illustrated

Table 3: Allocation results for each configuration per run.

| Tasks | Run 1, All Configs. | Run 2, Configs A and C. | Run 2, Configs B and D. | Both Runs, All Configs. |
|-------|---------------------|-------------------------|-------------------------|-------------------------|
| Allocated at the Edge | A | B | C | D | A | B | C | D |
| Allocated at the Hub | E | F | G | H | E | F | G | H |
| Allocated at the Cloud | I | J | K | L | I | J | K | L |

REFERENCES

[1] M. Akselrod, N. Becker, M. Fidler, and B. Luebben. 2017. 4G LTE on the Road—What Impacts Download Speeds Most?. In 2017 IEEE 86th Vehicular Technology Conference (VTC-Fall). 1–6. https://doi.org/10.1109/VTFCFall.2017.8288296
[2] Badrish Chandramouli, Joris Claessens, Suman Nath, Ivo Santos, and Wenchao Zhou. 2012. RACE: Real-time applications over cloud-edge. In Proceedings of the 2012 ACM SIGMID International Conference on Management of Data. ACM, 625–628.
[3] Thinh Quang Dinh, Jianhua Tang, Quang Duy La, and Tony Q. S. Quek. 2017. Offloading in mobile edge computing: Task allocation and computational frequency scaling. IEEE Transactions on Communications 65, 8 (2017), 3571–3584.
[4] D. Evans. 2011. The Internet of Things: How the Next Evolution of the Internet is Changing Everything. Cisco Internet Business Solutions Group (IBSG) 1 (2011), 1–11.
[5] Rajrup Ghosh and Yogesh Simmhan. 2018. Distributed Scheduling of Event Analytics Across Edge and Cloud. ACM Trans. Cyber-Phys. Syst. 2, Article 24 (July 2018), 28 pages. https://doi.org/10.1145/3140256
[6] LLC Gurobi Optimization. 2018. Gurobi Optimizer Reference Manual. (2018). http://www.gurobi.com
[7] Ying Kang, Johann Hauswold, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. SIGARCH Comput. Archit. News 45, 1 (April 2017), 615–629. https://doi.org/10.1145/3093337.3057698
[8] D. Kovachev, T. Yu, and R. Blamra. 2012. Adaptive Computation Offloading from Mobile Devices into the Cloud. In 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications. 784–791. https://doi.org/10.1109/ISPA.2012.115
[9] C. Kyckou, G. Pliorias, T. Theocharides, S. I. Venieris, and C. Bouganis. 2018. DroNet: Efficient convolutional neural network detector for real-time UAV applications. In 2018 Design, Automation Test in Europe Conference Exhibition (DATE). 967–972. https://doi.org/10.23919/DATE.2018.8342149
[10] Seng W. Loke. 2015. The Internet of Flying-Things: Opportunities and Challenges with Airborne Fog Computing and Mobile Cloud in the Clouds. CoRR abs/1507.04492 (2015). arXiv:1507.04492 http://arxiv.org/abs/1507.04492
[11] Xavi Masip-Brun, Eva Marin-Toerder, Ghazal Tashakor, Admela Jukan, and Guang-Jie Ren. 2016. Foggy clouds and cloudy fog: a real need for coordinated management of fog-to-cloud computing systems. IEEE Wireless Communications 23, 5 (2016), 120–128.
[12] S. Oudahourah, T. Tahel, J. Song, and C. Benzaid. 2017. Efficient offloading mechanism for UAVs-based value added services. In 2017 IEEE International Conference on Communications (ICC). 1–6. https://doi.org/10.1109/ICC.2017.7997362
[13] Matthew Ryden, Kwangsung Oh, Abhishek Chandra, and Jon Weissman. 2014. Nebula: Distributed edge cloud for data-intensive computing. In Collaboration Technologies and Systems (CTS), 2014 International Conference on. IEEE, 491–492.
[14] Vitor Barbosa Carlos de Souza, Wilson Ramirez, Xavier Masip-Brun, Eva Marin-Toerder, G Ren, and Ghazal Tashakor. 2016. Handling service allocation in combined fog-cloud scenarios. In Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA ’17). ACM, New York, NY, USA, 69–80. https://doi.org/10.1145/3079856.3080219