proceedings

MDPI

Proceedings

Optimal Assignment of Augmented Reality Tasks for Edge-Based Variable Infrastructures †

Angel Cañete *, Mercedes Amor and Lidia Fuentes

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, 29016 Málaga, Spain; pinilla@lcc.uma.es (M.A.); lff@lcc.uma.es (L.F.)

* Correspondence: angelcv@lcc.uma.es; Tel:+34-952-132-846

† Presented at the 13th International Conference on Ubiquitous Computing and Ambient Intelligence UCAmI 2019, Toledo, Spain, 2–5 December 2019.

Published: 20 November 2019

check for updates

Abstract: In the last few years, the number of devices connected to the Internet has increased considerably; so has the data interchanged between these devices and the Cloud, as well as energy consumption and the risk of network congestion. The problem can be alleviated by reducing communication between Internet-of-Things devices and the Cloud. Recent paradigms, such as Edge Computing and Fog Computing, propose to move data processing tasks from the Cloud to nearby devices to where data is produced or consumed. One of the main challenges of these paradigms is to cope with the heterogeneity of the infrastructures where tasks can be offloaded. This paper presents a solution for the optimal allocation of computational tasks to edge devices, with the aim of minimizing the energy consumption of the overall application. The heterogeneity is represented and managed by using Feature Models, widely employed in Software Product Lines. Given the application and infrastructure configurations, our Optimal Tasks Assignment Framework generates the optimal task allocation and resources assignment. The resultant deployment represents the most energy efficient configuration at load-time, without compromising the user experience. The scalability and energy saving of the approach are evaluated in the domain of augmented reality applications.

Keywords: edge computing; energy efficiency; Software Product Lines; Internet of Things; augmented reality

1. Introduction

The evolution of cyber–physical systems [1] and the Internet-of-Things (IoT) [2] is growing, fostering a massive use of devices ranging from smartphones, personal devices, home appliances, cloudlets and all kind of wearables sensors up to augmented reality glasses. Mobile Cloud Computing (MCC) [3], which integrates cloud computing and mobile computing, has empowered mobile devices with data storage, computational power and extra energy resources provided by the Cloud.

However, the full-scale spread of IoT devices contributed to the dramatic increase of the exchanged data, which directly affects to the network congestion, provoking the performance degradation of the purely cloud-based applications, and affecting user experience. Therefore, one of the main challenges faced by cyber–physical systems is to improve the user experience by reducing the user response time and power consumption, currently skyrocketed with the popularization of the MCC paradigm. As a response to this challenge, paradigms such as Edge Computing [4] and Fog Computing [5] were recently proposed.

Edge Computing is a computing paradigm that brings the advantages and power of the MCC [6] closer to where the data is created and consulted, by locating data, computation, storage and applications somewhere between the source of the data and the cloud. Concretely, Multi-access

Edge Computing (MEC, formerly known as mobile edge computing) [7] delegates some tasks to proximate computing entities. MEC takes advantage of the large amount of inactive computational power and distributed storage space in different edge devices (e.g., Wi-Fi routers) located nearby final users and connected to the same WLAN. The most important benefits of MEC are that it reduces the communication latency comparing with sending data to the cloud as in MCC, helps to decrease energy consumption due to communication and computation, and improves the data privacy. Exploiting the benefits that MEC and MCC can provide is not a trivial issue, due to the heterogeneity of the edge infrastructure and the mobile devices, and applications' demands. On the one hand, characteristics of the devices that can conform the nearby infrastructure (called *nodes*) are very diverse, in terms of computational power, RAM capacity, operating system, communication latency, etc. On the other hand, applications which are composed of a set of tasks can demand different QoS (Quality of Service) and QoE (Quality of Experience) capabilities, and also may require a specific device or resource. So, it can be a complex issue to find an appropriate assignment of tasks to nodes, fulfilling dependencies and requirements, before code offloading. Code offloading, which enhances the apparent capabilities of resource constrained devices [8], is interesting for developers trying to push ever more complex algorithms into embedded devices that belong to the IoT. However, the existing efforts and research directions made in the IoT domain show that they are not mature enough yet to accommodate the wide variability present in the IoT (standards, devices, network technologies, resources, capabilities) and still covers only a limited range of application scenarios [9].

In this paper, we present a solution for the optimal allocation of computational tasks to edge devices minimizing the overall energy consumption of the selected assignment. To manage the variability that both computational tasks and deployment infrastructures (DIs) we use *Feature Models* which are widely employed in *Software Product Lines*) [10]. The configuration of Feature models allow us to represent specific applications and deployment infrastructures, which are considered by the Optimal Tasks Assignment Framework (OTAF) to generate the optimal task allocation and resources assignment for a given application and infrastructure. The optimization problem is formalized and implemented in a SMT (Satisfiability Modulo Theory) [11] solver. The resultant deployment of the OTAF represents the most energy efficient configuration at loadtime without compromising the user experience. Although our approach can be used with any type of mobile applications, the work presented here is applied to augmented reality (AR) [12] applications. AR applications are composed by a considerable number of tasks, many of them computationally expensive and with specific hardware requirements, allowing us to face a very heterogeneity-rich scenario.There are several works and studies that show the benefits that edge computing can bring to AR applications; these demand low latency communications and real-time rendering and processing to offer the better user experience [13–15].

The rest of the paper is organized as follows: Section 2 introduces the background information to code offloading and poses its challenges. Section 3 presents our solution for information modeling and code offloading. Section 4 evaluates our approach in the domain of augmented reality applications, in terms of energy consumption and scalability. Finally, Section 5 presents the conclusions and the future work.

2. Background and Challenges

Code offloading is a technique commonly used to reduce the computation on the user's device. The initial idea behind code offloading in MCC is to transfer computations from a resource-limited mobile device to resource-rich cloud nodes to improve the performance of mobile application execution. Some of the potential benefits of code offloading are to shorten the execution time, achieve a reduction of energy consumption, avoid resource limitations of user devices that restrict the number of provided services, etc. Existing code offloading approaches differ in partition granularity, and device location, among other issues [16]. The granularity of the workload assigned to one device is defined as the different sizes of offloading components, and is variable. From the finer to the most coarse grained level, granularity for offloading use to be: at application level (binary offloading), at method level, at

component level or, the most commonly used, at task level (partial offloading). The more fine-grained the granularity, the more flexible and complex the offloading system is [16]. Offloading approaches also differ in the location of the devices to which a computation task is allocated. We can distinguish the proposals based on MCC, the approaches based on MEC, and hybrid approaches. In the proposals based on MCC, tasks are delegated exclusively to dedicated servers in the cloud [17,18]. In solutions based on MEC, tasks are allocated to the nodes of the nearby network infrastructure close to the final devices (namely, the edge of the network) [19,20]. Finally, there are solutions that combine the benefits of both paradigms (MCC + MEC), the so-called hybrid solutions [21,22], among which is our proposal. In this case, the computation tasks can be offloaded to both cloud and edge devices, benefiting from both proposals.

Another difference of code offloading approaches can be found in the strategy or algorithm used for the task assignment. The most popular methods include Markov decision processes [23], maximum weight matching [24], and integer programming [25]. As examples of integer programming approaches we can distinguish 0–1 linear programming (in which unknowns are binary), and mixed integer programming, where some decision variables are not discrete [26] (among which is our proposal). To address the potential requirements of future applications and the heterogeneity of the deployment nodes, a solution to model the deployment infrastructures and applications and an algorithm to use the information provided by them in order to achieve the benefits of MEC and MCC are necessary.

Challenges

The main goal of our approach is to provide the optimal task assignment, which is driven by the time required and the energy consumed associated with the data transmission and tasks computation for each device or node. To achieve this goal, our approach addresses the following challenges:

1. Manage the deployment infrastructure's variability: the nodes of the DIs have different hardware and software features (processing capacity, RAM, etc). At the same time, these devices are networked using different technologies with diverse capabilities.
2. Manage the applications' variability: the same type of application can have different configurations according to the functionality that the application must provide. It is necessary a mechanism to define the set of tasks that forms the applications from their configurations.
3. Comply with the applications' requirements: The partition of an application into tasks impose dependencies and constraints that must be fulfilled (sequential execution dependencies between tasks, and certain hardware or software requirements or software requirements). Also, non functional requirements involve factors such as the maximum time to complete a task or a set of them.
4. Provide the optimal assignment of tasks to the DIs: taking into account the factors of Challenge 3, the problem must be formalized and solved in order to perform an optimal assignment of tasks among the devices of the deployment infrastructure.
5. Optimize the assignment of infrastructure resources to users: If the current available resources of the DI are enough to allocate the applications' tasks, the allocation y distribution of the resources among the users must be optimized.

3. Our Approach

This section presents our approach and how it deals with the challenges that the optimal task assignment for computation offloading poses. Figure 1 depicts an overall view of the process. The approach is divided into the domain engineering and the application engineering. The domain engineering includes the feature models of applications and deployment infrastructures (Challenges 1 and 2). In general terms, feature models represent the information of all possible products (configurations) for a specific family of applications, modeling their features and the relationship between them. Feature models are represented as a set of hierarchically ordered features, composed

of parent-child relationships and a set of constraints (called *cross-tree constraints*). Our approach uses feature models to represent the applications and DIs separately. This separation allow us to reuse the DI feature model for different application families.

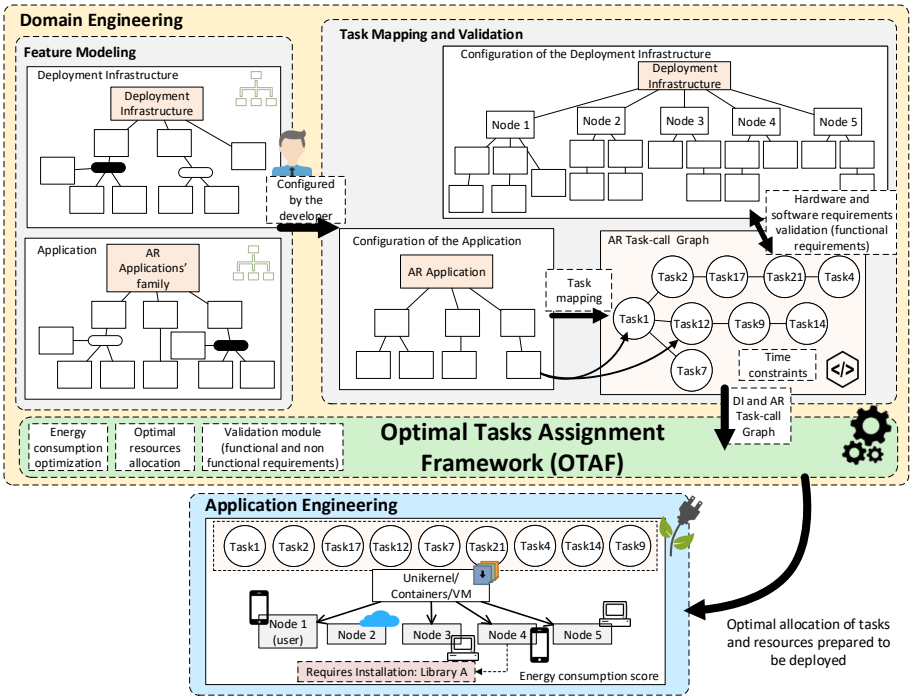

Figure 1. Overall view of the processes and models involved in our approach.

The application engineering is concerned with defining applications and DI by selecting and configuring the features they present. The configured (resolved) feature models are given to the Optimal Tasks Assignment Framework (OTAF), which is responsible for: (1) checking if it is possible to deploy the application on the DI; (2) deciding in which node of the DI will be deployed each task of the application; and (3) allocating hardware resources of the nodes to each task. The objective of the OTAF is to provide and assignment that minimizes the global energy consumption, accomplishing the functional and non functional requirements of the application.

These features models are used by the developer to configure both a specific application and the target DI. The configuration of the application's feature model is mapped with the set of tasks that form the application, which are modeled using *task-call* graphs [27].

3.1. DIs and AR Applications Features Modeling

Feature model of DIs: The infrastructures where mobile apps are deployed are composed of sets of nodes, which are characterized by: type of device, computing capacity, amount of memory, connectivity, software features, etc. The feature model of DIs (shown on top of Figure 2) collects the characteristics necessary to describe the nodes (addressing Challengue 3). Nodes have a type, a network to which they are connected and, optionally, a serie of sensors and an operating system. The nodes can be of type *computing* (destined to receive tasks offloaded by other users), *user node* (where the main task is executed) or *sensor mote*. Computers, mobile devices, Raspberry type computers and IoT Gateways (routers with processing capacity) that are located on the edge, as well as cloud devices, are computing nodes. The model contains different constraints, which make mandatory selecting a type and a O.S. Finally, it is also mandatory to select the network to which the node is connected. Technical information, such as the RAM, the CPU capacity and the network bandwidth are represented using feature models with numerical attributes. Since a DI can contain one or more similar nodes, we

use feature models with cardinality [28]. The cardinality in variability models allows the definition of the number of instances of a feature, in our case, the set of nodes in the DI. It is denoted by an interval $[n \ldots m]$, where n is the minimum number of occurrences of the feature and m is the maximum number.

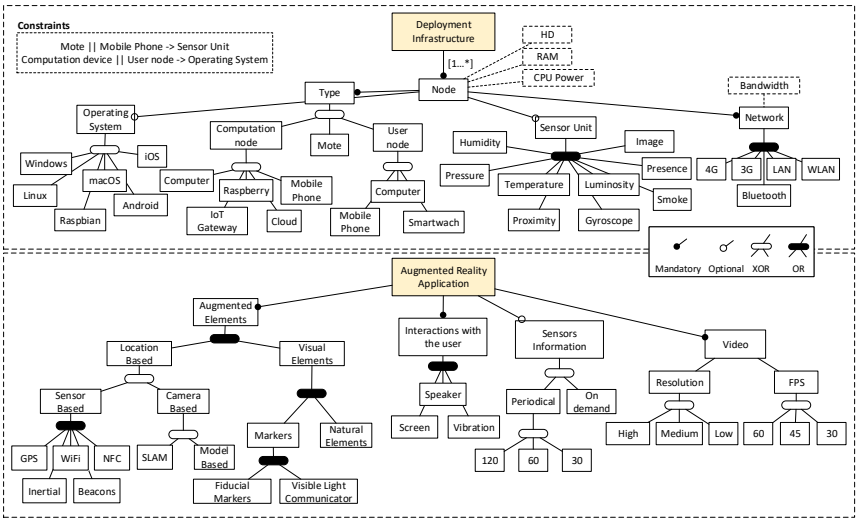

Figure 2. Feature models of a DI and a family of AR applications.

Feature model of the AR family of applications: a family of applications consists of a set of software applications that share most of their features. Feature modeling of a family of applications allows to configure and produce not only a specific application, but a group of them, as well as reuse software components. Feature models include all those characteristics that define the applications and their behavior.On bottom of Figure 2a feature model of the family of AR applications is shown. Augmented reality applications can be grouped by the elements that trigger the amplification of the reality (*Augmented Elements* in Figure 2). We distinguish AR applications based on location (e.g., Pokémon GO) and based on visual elements (e.g., fiducial markers). AR applications based on location, as their name suggests, enrich the reality according to the position of the devices. The location of the devices can be determined by using sensors or the device's camera. Sensor based approaches use elements like GPS (for outdoors), WiFi, NFC and beacons signals [29], or a set of them to resolve the location of the devices (e.g., inertial positioning [30] for indoors). The location can be determined using a camera by techniques like SLAM (*Simultaneous Location and Mapping*) [31], where the information obtained by the camera is used in order to create a map of the user's environment at runtime, or by the usage of model based techniques [32]. On the other hand, the AR applications based on visual elements trigger the augmented elements according to the current information obtained by the camera. Augmented elements can be artificial, like fiducial markers (e.g., QR codes) and visible light communicators [33], or natural (e.g., Instagram's filters based on face recognition).

Other AR features that can be configured in our model are the form in which the reality is augmented: by using the speaker of the user's node, by rendering virtual elements on the screen, or by the vibration of the device. The augmentation of the reality can be complemented with information collected by sensor motes. For instance, in a smart campus AR application, students would be able to know if teachers are in their office by a sensor in it. The information of the sensor motes can be consulted on demand (when required), or periodically. If the second option is chosen, the periodicity of the data collection (measured in seconds) can be setup. The feature model gathers the resolution and frames per second (FPS) supported by the AR app. The resolution determines the amount of data to be transmitted between the tasks responsible for image processing. The number of FPS is related with the maximum time required to identify the elements to be augmented with virtual objects.

3.2. Task Mapping and Validation

After the configuration of the feature models, the next step is to map the application's configured featured model to a task graph. In order to do this, the application configured feature model (which is in XML format) is processed and mapped with a set of know AR applications' tasks using the selected features by the developer. According to this configuration, the tasks may require more computational load or have to send more information to the rest. During the mapping process, it is checked if it is possible to deploy the given application on the deployment infrastructure modeled by the configured feature model. If not, the developer must reconfigure the feature models to include new nodes to the DI or change the application's functionality.

The task-call graph also includes information required to comply with application requirements (Challenge 3). Task-call graphs are composed of a set of vertices that represent the application's tasks (τ). These tasks contain tuples $t\ (w, m)$, where (w) is the associated computational load (measured in CPU cycles), and (m) the amount of RAM required by the task (measured in Mb). These values are commonly used for the representation of tasks and can be estimated [34]. The edges of the graph have information about the amount of data transmitted between tasks, measured in bits (Figure 3). If two tasks are not related, the weight of the link is 0. This allows to detect tasks whose beginning of execution depends on a previous task (sequential dependency) (e.g., the task *t2*, *t6* and *t13* in Figure 3), and the existence of parallelizable set of tasks (e.g., the tasks *t6* and *t5* in Figure 3). There are sets of tasks with time restrictions, with a maximum latency. These tasks are grouped into sets of tasks with sequential dependency, $r = \{t1, t2, t3, \ldots, tn\}$, with r_{time} as the maximum time to be completed. The set of these dependencies are grouped in the set R. These restrictions are specific to the application and must be fulfilled by all users. Figure 3 shows the task-call graph of an application with two time constraints: in the execution of the tasks *t6* and *t13*, (r_1), and in the execution of the tasks *t5*, *t18* and *t11* (r_2).

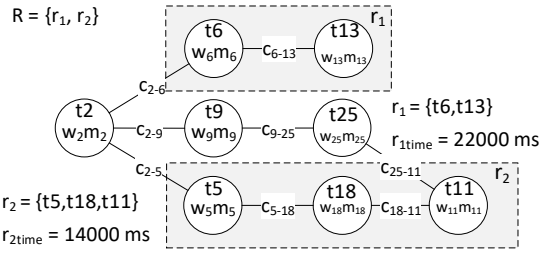

Figure 3. Graph representation of the tasks of an application.

3.3. Optimal Tasks Assignment Framework

The configured feature models of the DI and the application and the task-call graph are used by the OTAF to generate the optimal energy efficient deployment assignment. Firstly the XML file of the deployment infrastructure's configuration is processed to represent the necessary data to accomplish Challenges 4 and 5. The DI is formed by a set of nodes N. Each node is described as a set of hardware and network characteristics, modeled as a tuple $n(F, M, \kappa, C, P^{Tx}, ew)$. F represents the processing capacity, measured in clock cycles per second (Hz). M is the total RAM that the node can allocate. κ is a constant that depends on the hardware and that directly influences the energy consumption of computing a task [35]. C (bits/sec) represents the transmission capacity of the channel (determined by the Shannon Law [36]) and P^{Tx} (W) is the transmission power (which is assumed like constant [19]). These characteristics of the network to which the node is connected influence the transmission capacity of the channel, and can be calculated using power control mechanisms [37]. The nodes of the DI in which the energy consumption can be optimized are very diverse; user nodes, edge nodes, battery powered devices, all nodes, etc. By defining a system of policies based on nodes weighting, the nodes where the energy saving is activated, are highlighted. This is carried out by setting the value of n_{ew},

that establish the importance of energy saving in this node. For instance, if the goal is to minimize the energy consumption only in the user nodes, the energy weights of the rest of nodes are set to 0. The value of n_{ew} is a real number between 0 and 1. In our approach, we focus on scenarios in which the users' nodes are typically smartphones. For this reason, the battery level of the users' nodes is a key factor in order to select an energy policy.

Formalization of the Optimization Problem

The information structure presented in Sections 3.2 and 3.3 contains the information necessary to evaluate the computation and sending time, and the energy consumption in each node of the DI. The optimization function is given by the expression:

$$\begin{aligned} &\textbf{Minimize:} && \forall n \epsilon N : Esend_n + Eexec_n \\ &\textbf{Subject to:} && \forall t \epsilon \tau : \sum_{n \epsilon N} x_{t,n} = 1 && (1) \\ & && \forall n \epsilon N : \sum_{u \epsilon U} (m_{n,u}) \leq M_n && (2) \\ & && \forall r \epsilon R, n \epsilon N : \sum_{(i,j) \epsilon r} Texec_{i,n} + Tsend_{n,i,j} \leq r_{\text{time}} && (3) \end{aligned} \quad (1)$$

where $Esend_n$ is the energy consumption (J) in the node n due to data sending [19,38] and $Eexec_n$ is the energy consumption of a node due to tasks computation [35,39]; $Texec_{i,n}$ and $Tsend_{n,i,j}$ are the timing that the node n requires for executing its allocated tasks and data sending [19], respectively. The minimum power consumption solution is subject to a series of conditions: condition (1) verifies that each task is assigned to an unique node ($x_{t,n}$ is 1 if the task t is executed on node n, 0 if not). Restriction (2) checks that the sum of RAM resources allocated to users does not exceed the total memory resources of each node, accomplishing the Challenge 5 ($m_{n,u}$ is the RAM assigned to user u on node n). Finally, condition (3) verifies that, for each set of tasks with time constraints, the sum of the computation and sending time does not exceed the maximum time established to meet the QoS requirements (Challenge 3). Note that all these restrictions form the *Validation module* of the OTAF, as they check if it is possible to deploy the application on the DI according to the current DI's workload, accomplishing the application's requirements.

The RAM resources available in the nodes are distributed among the users (U) who use the application. Once these resources are assigned by the framework, they can be limited in practice using virtualization [40], containers [41] (e.g., Docker), or unikernel based solutions [42].

Once the OTAF is developed, its functionality is provided like a microservice based architecture. This service is running continuously on a node of the DI (*manager node*) and maintains updated information of each node of the DI (availability, workload, state of their resources, etc).

4. Evaluation

In this section, we evaluate our approach by configuring a DI, an AR application and then applying the OTAF to assign tasks of the resulting application to the DI's nodes. The energy consumption of the deployment assignment is compared with the execution of all the application's tasks at the user node in order to show the energy profit obtained in the user's device. Finally, the scalability of our approach is evaluated, modifying the size of the problem and measuring the response time from the OTAF.

Figure 4 shows our case study. Concretely, the figure is composed of three parts: (1) the top of the figure shows the configuration of the DI; (2) the second part shows the configuration of the application (using the feature model of Figure 2 in 1 and 2); (3) the bottom of Figure 4 shows the task-call graph of the application once processed from (2) (Section 3.2). The resulted application correspond to a SLAM AR application (Section 3.1), in which exist two set of tasks with time requirements: the tasks in charge of modify the current image obtained from a camera with the items to overlay (r_1) and the group of task that update the virtual map of the reality (r_2). Table 1 shows the policy selected (*we* values of the nodes) according to the battery level of the user's node used in this work.

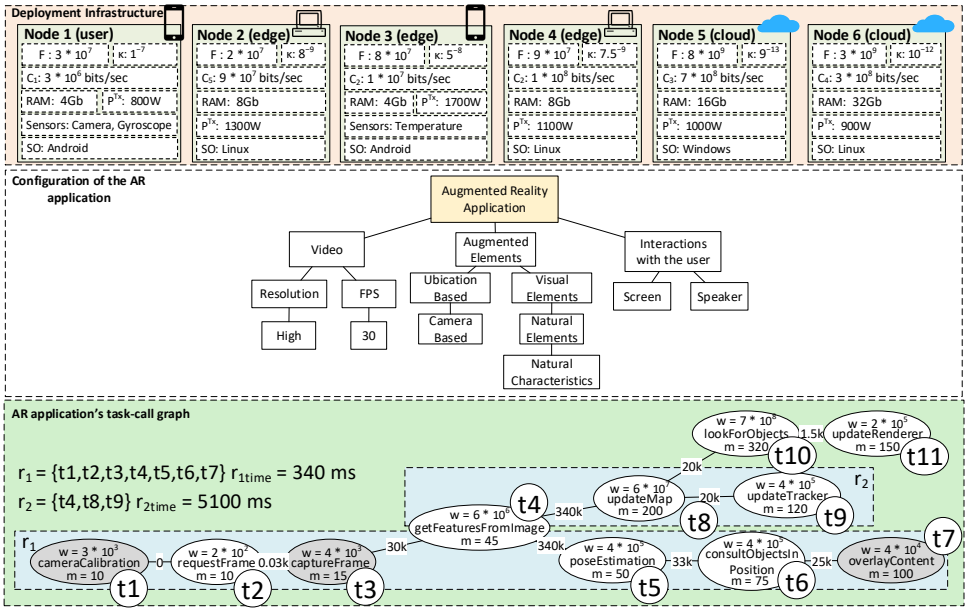

Figure 4. Case study of AR applications.

Table 1. Policy of energy weighting according to the battery level of the users' nodes.

User Node Battery Level (b)	Energy Weighting (n_{ew})		
	User Node	Edge Nodes	Cloud Nodes
100% $\leq$ b $\leq$ 75%	1.00	1.00	0.50
75% < b $\leq$ 50%	1.00	0.75	0.25
50% < b $\leq$ 25%	1.00	0.50	0.25
b < 25%	1.00	0.00	0.00

The optimization problem defined in Section 3.3 is implemented using a SMT solver. Specifically, we use the Z3 solver [43], an open source tool developed by Microsoft. In our case, the OTAF is executed in a node with an AMD Ryzen 7 1700X processor, running the service in one of its cores (Source code available at: http://caosd.lcc.uma.es/research/rsc/AR-Assignment.zip).

The assignment of tasks for the problem of Figure 4, using the values shown in the figure, is shown in the Table 2. In this case, the tasks *t1*, *t3*, *t4*, *t5*, *t6* and *t7* have been assigned to the Node 1 (user node), reserving 295 Mb of RAM for their execution. The tasks *t8*, *t9*, *t10* and *t11* have been assigned to the Node 2, reserving 10 Mb of RAM. Finally, the task *t2* has been assigned to the Node 6, keeping 10 Mb of RAM for its deployment. Note that nodes 3, 4 and 5 do not have assigned any task in this case. That is because for the available resources (all nodes are workload-free), this is the most energy efficient assignment according to our policy (Table 1). In this case, the battery level of the user node is 82%. The benefit in the energy consumption obtained in the user node respecting to running all application's tasks on it is 99.02%. The system has returned the solution in 1.4 s.

Table 2. Optimal tasks assignation for the problem of Figure 4.

Node	1	2	3	4	5	6
Tasks	t1, t3, t4, t5, t6, t7	t8, t9, t10, t11	-	-	-	t2
RAM assigned (Mb)	295	790	-	-	-	10
Battery level (user node)	**82%**					
Energy consumption's profit (user node)	**99.02%**					

Scalability

The process of assigning tasks is computationally expensive, since the number of allocation possibilities is N^{τ}. To measure the scalability of our proposal, we evaluate the time needed to obtain a solution by the framework for different sizes of the problem. To do this, we use a benchmark application that allows to select the number of nodes of the DI and the number of tasks that form the application (Source code available at: http://caosd.lcc.uma.es/research/rsc/OTAF-Benchmark.zip). The features of the application's tasks (e.g., CPU requirements) and nodes (e.g., CPU power, transmission power, type of node) are randomly generated by the benchmark application. Each experiment has been performed 20 times in order to show the average and standard deviation of the results. The battery level of the user node has been randomly generated too. For all the experiments, the system uses the policy shown in Table 1.

Number of tasks: the number of application's tasks is one of the most determinant factors in the size of the problem. In this experiment, we set the number of DI's nodes in 5 and increase the number of application's tasks up to 65 (left part of Figure 5). We see that for a fairly grainy application formed by 20 different tasks, the OTAF takes around 2.9 s to return an optimal allocation, which is acceptable by the user at loadtime. For very grainy applications formed by 60 tasks, the time goes up to 25.2 s.

Number of nodes: the number of devices that form the DI may affects the execution time of the framework. In this experiments, the number of application's tasks has been set in 10 and the number of nodes has been set up to 50 (right part of Figure 5). In this case, for a DI formed by 25 nodes, the framework requires around 8 s to return a solution.

In the both cases, the data dispersion show that there exist other factors that influence in the execution time of the framework, not only the number of tasks and nodes. This is due to several factors, like the complexity of the problem due to the features of the tasks and nodes. Other factor that we perceive during the realization of the experiments is the difference in time execution according to the policy. For the experiments in which the battery level in the user's node is under 25%, the OTAF provides a solution in a much shorter time than the rest (according to policy shown in Table 1). This is due to that in this case, the energy consumption of the both edge and cloud nodes are not taken into account, so the complexity of the problem decreases.

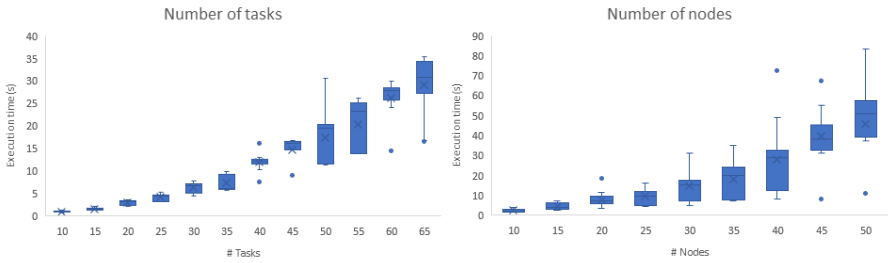

Figure 5. Time execution according to the number of tasks and nodes.

5. Conclusions and Future Work

As the number of devices connected to the Internet increases, the amount of data generated and consumed by these devices, which is stored and processed in the Cloud, can congest the network and affect negatively the energy consumption. Recent paradigms, such as Edge Computing and Fog Computing, propose to bring closer data processing from the Cloud to nearby devices in the Edge networks in order to alleviate this issue. Application functionality is divided into a set of tasks, which are offloaded to nearby nodes of the network infrastructure. Nevertheless, due to the variability of the applications that can benefit of MEC and network infrastructure's nodes, it is not easy to select the optimal deployment assignment for a DI and an application.

In this paper, we propose to use an SPL-based methodology to cope with the variability of deployment infrastructures and applications. We present the feature model of the DIs and the

feature model of the AR family of applications. The information obtained by the configuration of the feature models is processed and structured in order to be the input of the Optimal Tasks Assignment Framework, which returns the most energy efficient task assignment, accomplishing the functional and non functional requirements of the application in the process. In addition, the OTAF reserves the optimal amount of RAM resources in each node to users. To carry out these tasks, the assignment problem has been formalized as a optimization problem with restrictions and has been solved using a SMT solver. We developed a case study of an AR application, obtaining a high benefit in the energy consumption. Finally, the scalability of our proposal has been evaluated by measuring the execution time of the tasks assignment for different sizes of the problem, showing that the OTAF is able to return the optimal tasks' assignment in a time short enough to be provided at loadtime.

As future work, in addition to allocating RAM to users, we will assign CPU power in each node. We also propose to define the exchange of messages between the manager nodes and the rest of nodes of the DI in order to keep the updated information of the state of the DI at the manager node.

Author Contributions: Conceptualization, M.A. and L.F.; Methodology, M.A. and L.F.; Software, A.C.; Validation, A.C.; Formal Analysis, A.C.; Investigation, A.C.; Resources, L.C.; Data Curation, A.C.; Writing—Original Draft Preparation, A.C.; Writing—Review & Editing, M.A. and L.F.; Visualization, A.C.; Supervision, M.A. and L.F.; Project Administration, M.A. and L.F; Funding Acquisition, L.F.

Funding: This work is supported by the projects HADAS TIN2015-64841-R (co-funded by FEDER funds), TASOVA MCIU-AEI TIN2017-90644-REDT, MEDEA RTI2018-099213-B-I00 (co-funded by FEDER funds) and LEIA UMA18-FEDERJA-157 (co-funded by FEDER funds).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AR	Augmented Reality
DI	Deployment Infrastructure
FPS	Frames per Second
MCC	Mobile Cloud Computing
MEC	Mobile Edge Computing
OTAF	Optimal Tasks Assignment Framework
SMT	Satisfiability Modulo Theory
SPL	Software Product Lines

References

1. Lee, E.A. Cyber Physical Systems: Design Challenges. In Proceedings of the 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), Orlando, FL, USA, 5–7 May 2008; pp. 363–369.
2. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805.
3. Khan, A.u.R.; Othman, M.; Madani, S.A.; Khan, S.U. A Survey of Mobile Cloud Computing Application Models. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 393–413.
4. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646.
5. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and Its Role in the Internet of Things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC '12), Helsinki, Finland, 17 August 2012; ACM: New York, NY, USA, 2012; pp. 13–16.
6. Mell, P.; Grance, T. *The NIST Definition of Cloud Computing*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2011.
7. Porambage, P.; Okwuibe, J.; Liyanage, M.; Ylianttila, M.; Taleb, T. Survey on Multi-Access Edge Computing for Internet of Things Realization. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2961–2991.
8. Benedetto, J.I.; González, L.A.; Sanabria, P.; Neyem, A.; Navón, J. Towards a practical framework for code offloading in the Internet of Things. *Future Gener. Comput. Syst.* **2019**, *92*, 424–437.

9. Wang, Y.; Sheng, M.; Wang, X.; Wang, L.; Li, J. Mobile-Edge Computing: Partial Computation Offloading Using Dynamic Voltage Scaling. *IEEE Trans. Commun.* **2016**, *64*, 4268–4282. doi:10.1109/TCOMM.2016.2599530.
10. Pohl, K.; Böckle, G.; van Der Linden, F.J. *Software Product Line Engineering: Foundations, Principles and Techniques*; Springer Science & Business Media: Berlin, Germany, 2005.
11. Barrett, C.; Tinelli, C., Satisfiability Modulo Theories. In *Handbook of Model Checking*; Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R., Eds.; Springer International Publishing: Cham, Switzerlane, 2018; pp. 305–343.
12. Azuma, R.T. A Survey of Augmented Reality. *Presence Teleoper. Virtual Environ.* **1997**, *6*, 355–385.
13. Elazhary, H. Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions. *J. Netw. Comput. Appl.* **2019**, *128*, 105–140.
14. Ai, Y.; Peng, M.; Zhang, K. Edge computing technologies for Internet of Things: A primer. *Digit. Commun. Netw.* **2018**, *4*, 77–86. doi:10.1016/j.dcan.2017.07.001.
15. Bilal, K.; Khalid, O.; Erbad, A.; Khan, S.U. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Comput. Netw.* **2018**, *130*, 94–120. doi:10.1016/j.comnet.2017.10.002.
16. Wang, J.; Pan, J.; Esposito, F.; Calyam, P.; Yang, Z.; Mohapatra, P. Edge Cloud Offloading Algorithms: Issues, Methods, and Perspectives. *ACM Comput. Surv.* **2019**, *52*, 2:1–2:23.
17. Kristensen, M.D.; Bouvin, N.O. Scheduling and development support in the Scavenger cyber foraging system. *Pervasive Mob. Comput.* **2010**, *6*, 677–692. Special Issue PerCom 2010.
18. Lin, Y.; Chu, E.T..; Lai, Y.; Huang, T. Time-and-Energy-Aware Computation Offloading in Handheld Devices to Coprocessors and Clouds. *IEEE Syst. J.* **2015**, *9*, 393–405.
19. Dinh, T.Q.; Tang, J.; La, Q.D.; Quek, T.Q.S. Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling. *IEEE Trans. Commun.* **2017**, *65*, 3571–3584.
20. Verbelen, T.; Stevens, T.; Simoens, P.; Turck, F.D.; Dhoedt, B. Dynamic deployment and quality adaptation for mobile augmented reality applications. *J. Syst. Softw.* **2011**, *84*, 1871–1882.
21. Zhao, T.; Zhou, S.; Guo, X.; Zhao, Y.; Niu, Z. A Cooperative Scheduling Scheme of Local Cloud and Internet Cloud for Delay-Aware Mobile Cloud Computing. In Proceedings of the 2015 IEEE Globecom Workshops (GC Wkshps), San Diego, CA, USA, 6–10 December 2015.
22. Wang, S.; Urgaonkar, R.; Zafer, M.; He, T.; Chan, K.; Leung, K.K. Dynamic service migration in mobile edge-clouds. In Proceedings of the 2015 IFIP Networking Conference (IFIP Networking), Toulouse, France, 20–22 May 2015; pp. 1–9.
23. Papadimitriou, C.H.; Tsitsiklis, J.N. The Complexity of Markov Decision Processes. *Math. Oper. Res.* **1987**, *12*, 441–450.
24. Bayati, M.; Shah, D.; Sharma, M. Max-Product for Maximum Weight Matching: Convergence, Correctness, and LP Duality. *IEEE Trans. Inf. Theory* **2008**, *54*, 1241–1251.
25. Wolsey, L.A. *Integer Programming*; Wiley: Hoboken, NJ, USA, 1998.
26. Wolsey, L.A.; Nemhauser, G.L. *Integer and Combinatorial Optimization*; John Wiley & Sons: Hoboken, NJ, USA, 2014.
27. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358.
28. Czarnecki, K.; Helsen, S.; Eisenecker, U. Formalizing cardinality-based feature models and their specialization. *Softw. Process Improv. Pract.* **2005**, *10*, 7–29.
29. Liu, H.; Darabi, H.; Banerjee, P.; Liu, J. Survey of wireless indoor positioning techniques and systems. *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.* **2007**, *37*, 1067–1080.
30. Harle, R. A Survey of Indoor Inertial Positioning Systems for Pedestrians. *IEEE Commun. Surv. Tuto.* **2013**, *15*, 1281–1293.
31. Durrant-Whyte, H.; Bailey, T. Simultaneous localization and mapping: part I. *IEEE Robot. Autom. Mag.* **2006**, *13*, 99–110.
32. Reitmayr, G.; Drummond, T. Going out: Robust model-based tracking for outdoor augmented reality. In Proceedings of the Fifth IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR 2006), Santa Barbara, CA, USA, 22–25 October 2006; pp. 109–118.
33. O'Brien, D.C. Visible Light Communications: Challenges and potential. In Proceedings of the IEEE Photonic Society 24th Annual Meeting, Arlington, VA, USA, 9–13 October 2011; pp. 365–366.

34. Melendez, S.; McGarry, M.P. Computation offloading decisions for reducing completion time. In Proceedings of the 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2017; pp. 160–164.
35. Zhang, W.; Wen, Y.; Wu, D.O. Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In Proceedings of the 2013 Proceedings IEEE INFOCOM, Turin, Italy, 14–19 April 2013; pp. 190–194.
36. Wyner, A. Recent results in the Shannon theory. *IEEE Trans. Inf. Theory* **1974**, *20*, 2–10.
37. Xiao, M.; Shroff, N.B.; Chong, E.K.P. A Utility-based Power-control Scheme in Wireless Cellular Systems. *IEEE/ACM Trans. Netw.* **2003**, *11*, 210–221.
38. Mahmoodi, S.E.; Uma, R.N.; Subbalakshmi, K.P. Optimal Joint Scheduling and Cloud Offloading for Mobile Applications. *IEEE Trans. Cloud Comput.* **2018**, pp. 1–1.
39. Yuan, W.; Nahrstedt, K. Energy-efficient CPU Scheduling for Multimedia Applications. *ACM Trans. Comput. Syst.* **2006**, *24*, 292–331.
40. Lombardi, F.; Pietro, R.D. Secure virtualization for cloud computing. *J. Netw. Comput. Appl.* **2011**, *34*, 1113–1122.
41. Syed, M.H.; Fernandez, E.B. The Software Container Pattern. In Proceedings of the 22nd Conference on Pattern Languages of Programs (PLoP '15), Pittsburgh, PA, USA, 24–26 October 2015; pp. 15:1–15:7.
42. Bratterud, A.; Walla, A.; Haugerud, H.; Engelstad, P.E.; Begnum, K. IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services. In Proceedings of the 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), Vancouver, BC, Canada, 30 November–3 December 2015; pp. 250–257.
43. De Moura, L.; Bjørner, N. Z3: An Efficient SMT Solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*; Ramakrishnan, C.R., Rehof, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 337–340.