PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Universidade Federal do Rio de Janeiro

Igor Leão dos Santos

ON THE VIRTUALIZATION AND RESOURCE ALLOCATION IN THE CLOUD OF SENSORS

TESE DE DOUTORADO

Instituto de Matemática

Instituto Tércio Pacitti de Aplicações e Pesquisas Computacionais

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
INSTITUTO TÉRCIO PACITTI DE APLICAÇÕES E PESQUISAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Igor Leão dos Santos

ON THE VIRTUALIZATION AND RESOURCE ALLOCATION IN THE CLOUD OF SENSORS

Tese de Doutorado submetida ao Programa de Pós-graduação em Informática do Instituto de Matemática e do Instituo Tércio Pacitti da Universidade Federal do Rio de Janeiro (UFRJ), como parte dos requisitos necessários à obtenção do título de Doutor em Informática.

Orientadoras: Profª. Luci Pirmez, D.Sc.

Profª. Flávia Coimbra Delicato, D.Sc.

Rio de Janeiro
2017

S237 Santos, Igor Leão dos S..
On the Virtualization and Resource Allocation in the Cloud of Sensors / Igor Leão dos Santos. -- 2017.
122 f.: il.

Tese (Doutorado em Informática) – Universidade Federal do Rio de Janeiro, Programa de Pós-graduação em Informática, Instituto de Matemática, Instituto Tércio Pacitti, Rio de Janeiro, 2017.

Orientadoras: Luci Pirmez e Flávia Coimbra Delicato.

1. Cloud of Sensors. 2. Cloud Computing. 3. Edge Computing. 4. Information Fusion. 5. Virtualization. 6. Wireless Sensor and Actuator Networks. 7. Resource Allocation. 8. Optimization. 9. Mixed Integer Non-linear Programming. 10. Data Provisioning. 11. Data Freshness. 12. Heuristic Algorithm – Teses. I. Pirmez, Luci (Orient.). II Delicato, Flávia Coimbra (Orient.). III. Universidade Federal do Rio de Janeiro. Programa de Pós-graduação em Informática. Instituto de Matemática. Instituto Tércio Pacitti. IV. Título.

CDD:

ON THE VIRTUALIZATION AND RESOURCE ALLOCATION IN THE CLOUD OF SENSORS

Igor Leão dos Santos

Tese de Doutorado submetida ao Programa de Pós-graduação em Informática do Instituto de Matemática e do Instituo Tércio Pacitti da Universidade Federal do Rio de Janeiro - UFRJ, como parte dos requisitos necessários à obtenção do título de Doutor em Informática.

Aprovada em 05 de Dezembro de 2017 por:

Profª. Luci Pirmez, D. Sc., COPPE/UFRJ, Brasil (Presidente)
UFRJ/PPGI

Profª. Flávia Coimbra Delicato, D.Sc., COPPE/UFRJ, Brasil
UFRJ/PPGI

Prof. Paulo de Figueiredo Pires, D.Sc., COPPE/UFRJ, Brasil
UFRJ/PPGI

Prof. Luiz Fernando Rust da Costa Carmo, Docteur, UPS, França
UFRJ/PPGI

Prof. Célio Vinicius Neves de Albuquerque, Ph.D., UCI, EUA
UFF/IC

Prof. José Ferreira de Rezende, Docteur, UPMC, França
UFRJ/COPPE

I dedicate this thesis to my dear readers.
Especially to you who reads me now, in the future.
To you who either built your soul,
or still seeks self-knowledge.

Eu dedico essa tese aos meus caros leitores.
Especialmente a você quem me lê agora, no futuro.
A você que construiu sua alma,
ou ainda busca autoconhecimento.

Acknowledgment

I would like to sincerely thank everyone who supported this work somehow, including my family, especially my mother and aunts, who believed in my success during this work, and my friends that will forgive me of my absence in the last years.

My advisors, professors Luci Pirmez and Flavia Delicato who, with great wisdom and patience, guided me through all the steps for the conclusion of this work. Thank you professor Flavia for the several technical and grammar-related contributions to this work, and for all the insights and ideas that deeply enriched this work. Professor Luci, here follows a special and big thank you for all your help far beyond the role of an advisor. It has been more than a decade since I started learning from you several valuable lessons that I will take with me for life. You are the great example of success that inspired me to reach the end of this long road, as well as the person who showed me the path as carefully as a mother. Be sure that you can always count on my friendship at any time.

My coauthors that contributed in the publications obtained from this research. Especially, professors Paulo Pires and Luiz Rust from PPGI, professors Albert Zomaya and Wei Li from the University of Sydney, and professor Samee Khan from the North Dakota State University.

All the professors and secretaries of PPGI. Especially, professor Claudio Miceli who, besides being a coauthor of my works and being an inspiring example of dedication to the students and academia, also pioneered so many paths and left the accurate maps to me. And professor Priscila Machado Vieira Lima, for all the trust placed in me.

My colleagues and ex-colleagues (sorry, but you are too many, so I won't cite names) from the Laboratory of Wireless Networks and Multimedia (Labnet). In this free and scientific environment, during more than a decade we had several long talks and shared knowledge on the most heterogeneous topics (from computer science to health, humanities, sociology, politics and religion). I am sure that several of these talks have shaped the knowledge that will be transmitted by ourselves to the next generations. I thank you all for the opportunity to meet each one of you, for all the help with this thesis, for all the trust placed in me and for all the good, funny and happy moments we had together.

Finally, the CAPES foundation (Ministry of Education, Brazil) for the scholarship provided to me during the production of this thesis. I express my gratitude to the Brazilian government for believing in a project of nation that is strongly based on education, expanding the support to students in all levels during the last decades. Without this support, I would not have reached this high level of education and self-realization.

"When you use information from one source, It's plagiarism;
When you use information from many, It's information fusion".

– Dr. Belur V. Dasarathy
Editor-in-Chief
Information Fusion Journal (2013)

Resumo

SANTOS, Igor Leão dos S.. **On the Virtualization and Resource Allocation in the Cloud of Sensors**. 2017. 122 f. Tese (Doutorado em Informática) – Instituto de Matemática, Instituto Tércio Pacciti, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2017.

O novo paradigma da nuvem de sensores (CoS) vem ganhando visibilidade recentemente, reunindo os paradigmas de computação em nuvem e de redes de atuadores e sensores sem fio (RASSF) sob a forma de uma arquitetura de dois níveis. Com o surgimento mais recente do paradigma de computação de borda, disponibilizando recursos de computação, armazenamento e rede na borda da rede, tornou-se interessante considerar uma arquitetura de CoS de três camadas (compreendendo os níveis de sensor, borda e nuvem). Para proporcionar um nítido desacoplamento entre a arquitetura de três níveis de CoS e as aplicações, propomos o conceito de virtualização de CoS em nosso trabalho. Ao empregar a virtualização de CoS, um conjunto de nós virtuais (VNs) é disponibilizado para as aplicações. Alocar VNs às requisições de aplicações de maneira oportuna e eficiente, para atender aos requisitos das aplicações, dá origem ao desafio da alocação de recursos na CoS. Neste trabalho, formulamos o problema da alocação de recursos em CoS e propomos Zeus, um algoritmo parcialmente descentralizado para resolvê-lo. Escolhemos uma abordagem heurística para formular Zeus, devido à baixa sobrecarga de computação e rápida execução desse tipo de abordagem. Como características-chave, o Zeus é capaz de (i) executar requisições em comum entre múltiplas aplicações apenas uma vez, e compartilhar os resultados dessa execução única entre essas múltiplas aplicações, (ii) lidar com prioridades de aplicações e (iii) gerenciar relações de precedência entre as aplicações. Entre suas contribuições, Zeus é escalável em termos de número de VNs e aplicações na CoS, fornece suporte a aplicações sensíveis ao atraso e melhora a vida útil de WSANs.

Palavras-chave: Nuvem de Sensores, Computação em Nuvem, Computação em Borda, Fusão de Informação, Virtualização, Redes de Atuadores e Sensores Sem Fio, Alocação de Recursos, Otimização, Programação Não-Linear Inteira Mista, Provisionamento de Dados, Atualidade de dados, Algoritmo Heurístico.

Abstract

SANTOS, Igor Leão dos S.. **On the Virtualization and Resource Allocation in the Cloud of Sensors**. 2017. 122 f. Tese (Doutorado em Informática) – Instituto de Matemática, Instituto Tércio Pacciti, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2017.

The novel paradigm of cloud of sensors (CoS) gained momentum recently, bringing together the cloud computing and wireless sensor and actuator network (WSAN) paradigms in the form of a two-tier CoS architecture. With the more recent emergence of the edge computing paradigm, enabling computation and networking capabilities at the edge of the network, it became worth considering a three-tier CoS architecture (comprising the sensor, edge and cloud tiers). To provide a clean decoupling between the three-tier CoS architecture and applications, we propose the concept of CoS virtualization in our work. By employing such concept of CoS virtualization, a set of virtual nodes (VNs) is made available to applications. Assigning VNs to application requests in a timely and efficient way, in order to meet the requirements of applications, gives rise to the challenge of resource allocation in CoS. In this work, we formulate the problem of resource allocation in CoS and propose Zeus, a partly decentralized algorithm for solving it. We choose a heuristic approach to formulate Zeus, due to its low computation overhead and fast execution. As its key features, Zeus is capable of (i) performing requests in common for multiple applications only once, sharing the results of this single execution among these multiple applications, (ii) handling priorities and (iii) handling precedence relationships among applications. Among its contributions, Zeus is scalable in terms of the number of VNs and applications in the CoS, provides support to delay-sensitive applications, and improves the lifetime of WSANs.

Keywords: Cloud of Sensors, Cloud Computing, Edge Computing, Information Fusion, Virtualization, Wireless Sensor and Actuator Networks, Resource Allocation, Optimization, Mixed Integer Non-linear Programming, Data Provisioning, Data Freshness, Heuristic Algorithm

List of Figures

List of Tables

List of Acronyms

AEP	Application Entry Point
AMS	Application Manager Subsystem
AMSA	Average Makespan of Applications
ANUR	Avoid Negative Utility Rule
API	Application Programming Interface
CDCs	Cooperative Damage Coefficients
CMS	CoS Manager Subsystem
CN	Cloud Nodes
CoS	Cloud of Sensors
CRAM	Centralized Resource Allocation Manager
CRPM	Centralized Resource Provisioning Manager
CSCs	Cloud Service Customers
CT	Cloud Tier
DAG	Directed Acyclic Graph
DAI-DAO	Data In-Data Out
DAI-DEO	Data In-Decision Out
DAI-FEO	Data In-Feature Out
DEI-DEO	Decision In-Decision Out
DFD	Data-Feature-Decision
DPM	Data Provisioning Manager
DRAM	Decentralized Resource Allocation Manager
DRPM	Decentralized Resource Provisioning Manager
DSL	Domain Specific Languages
DSM	Data Storage Manager
DT	Data Type
DUpTm	Data Update Time
EB	Event-Based
EBR	Energy Balancing Rule
ENORM	Edge Node Resource Management
EN	Edge Nodes
ET	Edge Tier
FEI-DEO	Feature In-Decision Out
FEI-FEO	Feature In-Feature Out
FIFO	First In–First Out
GEs	Generic Enablers
ICT	Information and Communication Technologies
InPs	Infrastructure Providers
IoT	Internet of Things
LAN	Local Area Network
LTST	Lifetime Of Sensors Tier
MINPP	Mixed Integer Non-linear Programming Problem
MUR	Maximum Utility Rule
PAC	Percentage of Applications Completed
PLA	Percentage of Late Applications

PSAN	Physical Sensor and Actuator Nodes
QoM	Quality of Monitoring
QTRR	Queue Time Reduction Rule
rq	Request
RR	Registries Repository
SAM	Sensing and Actuation Manager
SensorML	Sensor Modeling Language
SLA	Service Level Agreement
SSAN	Shared Sensor and Actuator Networks
ST	Sensors Tier
TB	Time-Based
TECCT	Total Energy Consumption By The Cloud Tier
TECET	Total Energy Consumption By The Edge Tier
TECST	Total Energy Consumption By The Sensors Tier
TUTOB	Total Utility Obtained for Applications
UDAG	Unique DAG
urq	Unique Request
VMs	Virtual Machines
VN	Virtual Node
VNIR	VN Instances Repository
VNM	VN Manager
VNS	Virtual Node Subsystem
VS	Virtualization Subsystem
VSM	Virtualization Subsystem Manager
WSAN	Wireless Sensor and Actuator Network
WSANaaS	WSAN as a Service
WSANI	Wireless Sensor and Actuator Network Infrastructure

Summary

1 Introduction

During the first decades of the XXI century, several sectors of industry, academia and society have been witnessing the wide spreading of a multitude of new paradigms, which are revolutionizing the field of Information and Communication Technologies (ICT) [34]. The novel cloud of sensors (CoS) paradigm [40] is one of such paradigms that has gained momentum recently. The CoS combines wireless sensor and actuator networks (WSANs) [26] with cloud computing [6], making up a two-tier architecture.

Formally, a cloud is a large-scale (ideally unlimited) set of virtualized computing resources that owners can dynamically reconfigure to serve a variable load, seeking optimum resource utilization [6], [33]. Accordingly, virtualization is a key feature that grounds the cloud computing paradigm. The authors in [33] define virtualization in terms of hiding from clients the variety of types of infrastructures, platforms and data available at the back-end, facilitating application delivery.

Recently, another paradigm built on virtualization principles has been gaining momentum: the edge computing [39]. Edge computing proposes the virtualization of physical edge devices, to enable computation, storage, and networking capabilities at the edge of the network, i.e. among the sensor nodes and traditional clouds [39]. Physical edge devices are heterogeneous in terms of their capabilities and can be either resource-poor devices such as access points, routers, switches, base stations, and smart sensors, or resource-rich machines like a “cloud-in-a-box”, such as Cloudlets [36]. Edge devices perform a number of tasks. For instance, collecting the data and performing preprocessing, filtering the data and reconstructing it into a more useful form, uploading only the necessary data to the cloud. In addition, edge nodes can monitor smart objects and sensors activities, keeping check on their energy consumption. The edge consumes locally the portion of data generated by sensors that require real-time processing (from milliseconds to tenths of seconds). Then, it transmits the rest of such data to the cloud, for operations with less stringent time constraints (from seconds to minutes) [39]. Therefore, edge computing allows real-time delivery of data, especially for latency sensitive services. On the other hand, the closer to the cloud, the longer the time scale, and the wider is the geographical coverage. The cloud provides the ultimate

and global coverage, and serves as a repository for data for a duration of months or years, besides allowing more complex data analytics, based on historical data.

Combining WSANs and cloud/edge computing paradigms in the form of a three-tier CoS architecture (comprising the sensor, edge and cloud tiers) potentially leverages mutual advantages. On the one hand, WSANs could benefit from the virtually unlimited resources, besides the low latency, mobility and location-awareness support, of a cloud/edge environment to implement service management and composition for exploiting the smart sensors and their produced data. On the other hand, the cloud/edge computing paradigms can benefit from WSANs by extending their scopes to deal with real world objects in a distributed and dynamic way, enabling the delivery of a wider variety of new services in real world scenarios. However, despite the advantages of the three-tier CoS architecture, one important challenge in the design of the CoS pertains to the development of a model for CoS virtualization, which we describe in Section 1.1.

1.1 The challenge of CoS virtualization

Essentially, in the CoS paradigm, the cloud and edge tiers perform CoS virtualization, which is built on the concept of WSAN virtualization [7]. The concept of CoS virtualization provides a clean decoupling between the whole three-tier CoS infrastructure and applications via an abstract representation of the data, computation and communication capabilities of the former. Through CoS virtualization, it is possible to hide from users the complexity of the three-tier CoS infrastructure, facilitating application delivery and allowing the sharing of such infrastructure among several applications.

Traditional two-tier CoS architectures that are proposed in the literature [3], [4] consider the physical sensors as passive devices able to provide data to the closest sink node, which forwards such data to a (often) single database stored in the Cloud. Being fully inside the Cloud, the CoS virtualization takes place, simply based on processing/correlating data stored in this database, and thus in a centralized manner. Such a model is traditionally known as Sensing as a Service [3]. Moreover, CoS virtualization is usually based on publish/subscribe mechanisms [6], where each physical sensor publishes the sensed data and metadata (comprising sensor types, locations, and other useful descriptive information), and applications subscribe to the published sensor data. Each application subscription to a published set of sensor data results in the instantiation of new virtual nodes, or reuse of

existing ones. Consequently, the instances of virtual nodes are created and exist only inside the Cloud, based on the (possibly correlated) data provided by the existing physical sensors connected to the CoS. These centralized CoS virtualization approaches demand transmission of data to a sink node connected to the WSAN. The communication overhead caused by such an approach is aggravated when large-scale physical deployments are used to increase the frequency of simultaneous transmissions. This communication overhead results in at least two major drawbacks that hinder centralized CoS infrastructures to achieve better results when used as solutions to the challenge of the CoS virtualization.

The first major drawback is the compromising of the WSAN lifespan, as the nodes of the WSAN have limited energy resources [3], [9], [11]. An effective solution for maximizing the system lifespan is to process and reduce the sensed data locally, within the physical WSAN. Such data reduction consists on decreasing the amount of data (that reduces the corresponding transmissions) used by applications to make decisions. A possible approach for data reduction is to make use of Information Fusion [9], which consists of the transforming/joining (fusing) of two or more pieces of information (data) from different sources, resulting in other information. In this approach, it is possible to consider virtual nodes as computational entities capable of performing a set of information fusion techniques. Therefore, it is possible to map the virtualization of physical sensors onto the three data abstraction levels of information fusion (measurement, feature and decision) [9], based on the input and output data of each of the virtual node.

The second major drawback being that such a communication overhead imposes an additional delay on top of the response time of applications running within the CoS. The response time of applications comprises of the execution time of data acquisition, processing, decision, and actuation. In centralized CoS infrastructures [3], [6], [8], the response time of applications depend on several factors, such as: (i) the communication bottlenecks formed close to the sink nodes, (ii) the size (in hops) of the physical WSAN, (iii) the delay of routing data inside the Cloud, (iv) the delay in processing and making decisions within the Cloud, and (v) the delay of issuing an actuation message back to the physical WSAN. Although the approach of data reduction aids in decreasing the time spent due to each of the above factors, when transmitting the data via a sink node to the Cloud for instantiating virtual nodes and data processing, the total response time still comprises of all such factors. Such a situation

impedes several applications that require fast response [10], [18]. To overcome such a restriction, a feasible approach is to decentralize the decision processes of applications [11], [22], [40]. That is to say that the decentralized approach consists on performing the decision processes of applications inside the physical sensor, leveraging the in-network processing capabilities of the nodes.

Further on this account, the existing publish/subscribe models proposed within the centralized CoS infrastructures ignore that sensors also have local processing and communication capabilities for performing localized and collaborative algorithms, which are required for applications that are inherently decentralized [11], [22], [40]. In particular, the adoption of the WSAN as one of the cornerstones of the Internet of Things (IoT) paradigm fosters the introduction of novel and more complex applications, such as domotics, assisted living, e-health, business/process management, structural health monitoring, and intelligent transportation of people and goods. To complete complex tasks in the IoT scenario, the applications require distributed processing within the network. In general, the centralized CoS infrastructure approach is unsuitable for the execution of decentralized applications. Therefore, for supporting a broader set of applications, it becomes essential that the CoS infrastructure allows the execution of localized and collaborative algorithms as a service within the physical sensors. Such an approach leads to a novel paradigm called the WSAN as a Service (WSANaaS), in which the concept behind the services provided by a WSAN node is much broader than the concept proposed in the traditional Sensing as a Service paradigm [3].

Therefore, the challenge of CoS virtualization refers to the development of a model for CoS virtualization that simultaneously meets the requirements of several applications, while dealing with the resource constrained nature of the WSANs and prolonging their lifespan. Furthermore, by employing CoS virtualization, a set of virtual nodes (VNs) is made available to applications. The need to assign VNs to application requests in a timely and efficient way gives rise to the challenge of resource allocation in CoS. We describe the challenge of resource allocation in CoS in Section 1.2.

1.2 The challenge of resource allocation in CoS

The goal of resource allocation in CoS is to maximize the amount of application requirements properly met by the CoS infrastructure, while ensuring a target operational cost [41]. The resource allocation in CoS differs from the traditional cloud computing approach by

dealing with the specificities of CoS systems. For instance, in traditional cloud computing, the sharing of computational, storage and networking capabilities of cloud servers among cloud applications is the main goal. In contrast, in CoS, the sensor data acquisition and sharing among CoS applications is a more important goal than sharing the computational, storage and networking capabilities of devices from either sensors, edge or cloud tiers. Therefore, the CoS systems are specifically based on sensing data, and so we consider resource allocation in CoS under the prism of data provisioning [42], [43]. The philosophy behind data provisioning consists of abstracting, to users and applications, data acquisition/access, and allowing data sharing among applications, while meeting respective application requirements. Among the possible requirements stands out the data freshness, given its importance for distributed systems based on sensing data. There are several definitions of data freshness in literature [13]. Among these definitions, throughout this work we will adopt the one that quantifies the freshness of a given data based on the time elapsed since its acquisition. In our approach based on data provisioning, the resource allocation problem links the data provided by VNs, provided by the CoS virtualization model, and the data demanded by CoS applications. Hereafter, we describe the main aspects of VNs and CoS applications that we consider in the formulation of the problem of resource allocation in CoS.

The concept of virtual nodes (VNs) denotes an abstract representation of (i) data, (ii) computation and (iii) communication capabilities of the CoS infrastructure. Such VNs are computational entities implemented as software instances that run on top of the edge or cloud tiers. Therefore, we consider that VNs are analogous to typical IoT resources [44], [76]. Each VN exposes a data provisioning service, which provides data in response to application requests. Moreover, since we adopt a virtualization approach based on information fusion, each VN represents an information fusion technique [40], [9]. Therefore, the VN is capable of coordinating the underlying CoS infrastructure for performing the sensing, processing or actuation necessary to accomplish the execution of the information fusion technique represented by it. The data provided by the data provisioning service of each VN is the output of its implemented information fusion technique. Such output data may comprise raw sensing data, processed sensing data or even control actions, in case of VNs that abstract actuation tasks. Moreover, the output data of VNs is of a single data type.

In our work, each data type is unique, and several VNs can provide data of the same data type. The CoS Infrastructure Providers (InPs) define and describe the data types available in the CoS system. For instance, in the context of a SHM application, a data type D can be a structural damage index calculated through a given damage detection technique [22]. In addition, InPs can also define the data types R1 and R2 as the raw data, obtained from two different civil structures (1 and 2, respectively), used in the calculation of D. Users define their applications based on the data types available in the CoS. Therefore, we consider the possibility of supporting users from all levels of expertise in application domains. The InPs are responsible for considering the desired level of expertise of their target users, when defining the data types. For instance, users from a higher level of expertise can be interested in building applications using D and R1, or D and R2. In other words, the VN that provides D can receive data from either VNs that provide R1 or R2. However, users from a lower level of expertise can be interested in building applications using a data types that abstract data acquisition. For instance, they might request data types D1 and D2, which represent a damage index calculated for civil structures 1 and 2, respectively.

Regarding CoS applications, we inspire our application model in a workflow based approach, from the area of Web Services [30], [31]. Users model an application as a workflow that comprises several activities. In this work, we call each activity of an application workflow as a request. A request is a logical interface to the data provisioning service implemented by VNs. Therefore, requests do not hold any implementation of services. In addition, for each request, users define a set of requirements. We consider two categories of requirements: non-negotiable, which must be completely fulfilled (100% fulfillment level), and negotiable, which allow any given level of partial fulfillment, under 100% [15].

In our work, we consider the data type as a non-negotiable requirement. Several VNs that provide the same data type are alternatives to meet a request. We consider the data freshness as a negotiable requirement. Ideally, each VN should update its data upon its allocation to each application request, in order to meet all requests with best (zero) data freshness. However, data updates require the VN to coordinate the engagement of the underlying CoS infrastructure, thus incurring in a given processing load on it. Besides consuming energy/bandwidth resources, the execution of data update procedures generate a delay for meeting the request. To avoid the constant re-execution of processing loads by the

underlying CoS infrastructure, it is possible to meet requests by simply re-using data outputs stored locally within the VN. This data is stored from previous data updates and can be reused, provided that such data meets the data freshness requirement of the requests. Thus, only requests demanding fresher data than the one currently stored in a VN will require data updates, minimizing the system overhead.

Therefore, to achieve its goal, the resource allocation process in CoS comprises three simultaneous decisions. The first one regards which VN should meet each request. The second decision regards when (in which moment in time) the VN should meet each request. The third decision regards if each VN should update its data to meet the request. This resource allocation process is a fundamental part of our three-tier CoS architecture, which also has several other components that perform functions required for the full operation of a CoS system.

As in other works in literature [73], the problem of resource allocation in CoS falls within the typical class of mixed integer non-linear programming problems (MINPP) [17], [66]. To solve our MINPP in order to seek the optimal solution, there are a number of methods in the literature, such as the linear programming techniques and their variants [67]. However, our formulated MINPP is an NP-complete problem [46], so its explosive combinatorial nature hinders the quick search for optimal solutions when it grows, in terms of the number of VNs and applications. This aspect harms the typical delay-sensitive applications from the scenario of edge computing, which usually require strict response times, in the order of milliseconds to sub seconds [18]. In this sense, another challenge arises regarding how to solve, in polynomial time, practical instances of our MINPP with arbitrary sizes [78]. Because of NP-complete problems, an entire research area exists that deals with quick search of solutions, emphasizing heuristic techniques that produce near-optimal solutions, and also show low computation overhead [68], [77], [85]. Among the several classifications of heuristic techniques shown in [85], we chose to use constructive techniques for the following reasons. First, they do not require initial solutions, allowing the decision-making node that uses these constructive techniques to be more autonomous, in terms of requiring less inputs to make decisions, than other techniques shown in [85]. In addition, constructive techniques do not require the decision-making node to have information about the whole scenario to make decisions.

Instead, it can make localized decisions, with the information available locally, fostering decentralized decision-making.

1.3 Research questions

Based on the abovementioned challenges of (i) CoS virtualization and (ii) resource allocation in CoS, we investigate the following research questions in our work.

Research question 1: *In the context of the Cloud of Sensors paradigm, the concept of virtualization is built on the concepts of Resource Provisioning, Resource Allocation and Task Scheduling.*

To be fully accomplished, a WSAN virtualization scheme in CoS comprises at least three sub processes, each one for implementing Resource Provisioning, Resource Allocation and Task Scheduling. Typical approaches for these processes already exist in traditional areas related to the field of CoS, such as Cloud Computing [94][41], and WSAN [24]. In the field of CoS, new requirements arise for these three processes. For instance, the CoS is a typical distributed system based on sensing data, where data integration is an important issue. Thus, data freshness is an important requirement for the CoS. Therefore, in order to propose a CoS virtualization model, it is important to investigate solutions to these three processes that consider the specific requirements from the context of CoS.

Research question 2: *A CoS virtualization model that is based on information fusion is able to reduce data transmissions and, consequently, to save energy to the CoS system, extending the lifetime of WSANs. Due to decentralizing applications' decision processes, such a CoS virtualization model is also able to reduce the response time of applications.*

There are several drawbacks to a centralized CoS virtualization approach. The first, communication overhead, compromises the WSAN lifespan, since the nodes of the WSAN have limited energy resources. An effective solution for maximizing the system lifespan is to process and reduce the sensed data locally, within the physical WSAN. Such data reduction consists of decreasing the amount of data (which reduces the corresponding transmissions) used by applications to make decisions. A possible approach for data reduction is to use information fusion, which consists of transforming/joining (fusing) two or more pieces of information (data) from different sources, resulting in other information.

The second major drawback is that such a communication overhead imposes an additional delay on top of the response time of applications running within the CoS.

Application response time consists of the execution time of data acquisition, processing, decision, and actuation. In centralized CoS infrastructures, an application's response time depends on several factors [6][3][8], such as (i) the formation of communication bottlenecks close to the sink nodes, (ii) the size (in hops) of the physical WSAN, (iii) the delay in routing data inside the cloud, (iv) the delay in processing and making decisions within the cloud, and (v) the delay in issuing an actuation message back to the physical WSAN.

Although the data reduction approach helps lessen the time spent due to each of these factors, when transmitting the data via a sink node to the cloud for instantiating virtual nodes and data processing, the total response time still comprises all such factors. Such a situation impedes several applications that require fast response [10]. To overcome this restriction, a feasible approach is to decentralize applications' decision processes—that is, perform application decision processes inside the physical sensor, leveraging the nodes' in-network processing capabilities [11].

Research question 3: *A hybrid and heuristic-based algorithm to perform resource allocation in CoS, which considers precedence relationships among application requests and shares the results of requests in common among applications, is (i) scalable, (ii) able to support delay-sensitive applications and (iii) able to save energy from WSANs, improving their lifetime.*

1.4 Objective

This thesis has two distinct goals. Based on research questions 1 and 2, the **first goal** of this work is to propose a novel information fusion and decentralized CoS virtualization model, which we term Olympus. Olympus seeks to make the best use of both the Cloud and the physical WSAN environments, by finding a balance between two possible approaches for running services: (i) centrally, inside the Cloud and (ii) locally, within the physical sensors. Compared to the state of the art, the main distinct features of Olympus are:

First, Olympus builds on the key concept of information fusion since each of the virtual nodes represents the execution of an information fusion technique. This ensures the system to provide data at a given abstraction level [9] during the information fusion process.

Second, Olympus is a decentralized CoS virtualization model because physical nodes can perform locally the necessary procedures for creating and running the virtual node. The virtual node creation and operation management is not fully held centrally, within the Cloud.

Therefore, in Olympus, the decision processes of applications are performed partly within physical sensors and partly within the Cloud.

Based on research question 3, the **second goal** of this work is to propose Zeus, a heuristic-based and hybrid algorithm to solve the MINPP for resource allocation in CoS. Zeus is designed to run on a CoS scenario grounded on the concepts of our proposed CoS virtualization model, Olympus. In such a CoS scenario of large dimensions and serving multiple applications, our algorithm has the following key features, which inherit the previously mentioned features of Olympus.

First, in Zeus we adopted a heuristic approach for finding near-optimal solutions to maximize the data freshness provided to applications. In such a data freshness-based approach, Zeus is able to reuse data obtained in previous moments in time to meet current applications, considering their respective negotiable data-freshness requirements. To the best of our knowledge, there is no other work that proposes a formulation to the problem of resource allocation in CoS that considers the data freshness requirement.

Second, Zeus is capable of performing requests in common for multiple applications only once, sharing the results of this single execution among these multiple applications. Therefore, besides reusing data obtained in different moments in time, as stated by the first feature, Zeus is also capable of reusing the same data to meet multiple applications simultaneously in time.

Third, it considers the existence of precedence relationships (dependencies between data inputs and outputs) among the requests of a same application. Therefore, VNs must provide data in first place to the requests whose inputs are satisfied (all the preceding requests completed), and are free to start being processed.

Fourth, we leverage the concept of edge computing in Zeus operation. By doing so, the resource allocation process is not limited to occur only in the cloud tier, as in two-tier CoS architectures [3]. Instead, our design allows Zeus to run at the edge tier, and considers that the instantiation of VNs occurs at the edge tier.

Fifth, Zeus is designed as a hybrid (partly-decentralized) algorithm. In centralized resource allocation algorithms for CoS [71][46][49], the decision is taken by a centralized entity, and for a set of passive VNs, i.e. VNs do not partake in the decision process. Therefore, such algorithms are usually implemented fully within the cloud, considering the global view of

the CoS environment, i.e. considering all VNs in CoS. In turn, in decentralized algorithms [70], the resource allocation decision is taken cooperatively among VNs and independently from a centralized entity. Therefore, such algorithms are usually implemented fully within the VNs, which determine their allocations to requests based on their local views. In Zeus, we share characteristics of both centralized and decentralized algorithms. We combine a centralized decision phase, which runs in either cloud or edge tiers, and a decentralized decision phase, held fully within VNs. Therefore, the hybrid design of Zeus makes the most of the features of each computational tier of the CoS system.

1.5 Contributions

The major goals of this thesis regard proposing a novel CoS virtualization model, Olympus, along with a new algorithm to perform resource allocation in CoS, Zeus. In this Section, we point out and discuss the major contributions of our approach, derived from the key features of Zeus, which inherit the features of Olympus.

As our first contribution, Zeus is scalable, both in terms of the number of VNs and the number of applications executed in the CoS. This contribution is a consequence of the hybrid approach of Zeus, which would not be possible without the support of a decentralized virtualization model as Olympus. Olympus contributes for enabling the decentralization of the WSAN virtualization process, and by supporting both centralized and decentralized applications simultaneously within the CoS infrastructure.

As our second contribution, Zeus provides support to delay-sensitive applications. This contribution is achieved by leveraging edge tier, what allows supporting delay-sensitive applications (three-tier approach), in comparison to an approach using only the cloud tier (two-tier approach). Once again, Olympus plays a key role to achieve this contribution, since it considers the edge tier, allowing the design of Zeus for the three-tier CoS. This contribution is also a result of the use of both the use of information fusion and the decentralization of the WSAN virtualization process considered by Olympus, which reduce the response time of applications, by reducing communication overhead in CoS. Moreover, Zeus is capable of finding solutions in reduced computation time to the problem of resource allocation in CoS, also contributing to support delay-sensitive applications.

As our third contribution, Zeus saves energy and consequently improves the lifetime of WSANs. This contribution is a direct consequence of using Zeus mechanism for identifying

tasks that are common for multiple applications, performing them only once and sharing the outcome among these multiple applications. Moreover, it is important to mention that this contribution would not be possible without considering the key features of Olympus that also contribute to the improvement of the lifetime of WSANs. In this case, the key features are the use of information fusion techniques for reducing data transmission, and the decentralization of WSAN virtualization procedures. We also explicited the trade-off between the quality of solutions found by Zeus and the respective amount of energy consumed by WSANs to achieve them. We showed how much energy can be saved for the WSANs when adopting the approach of reusing data obtained in different moments in time to meet applications, while considering their negotiable data freshness requirements.

1.6 Organization

Hereafter, we organize this thesis as follows. Chapter 2 discusses the CoS architecture and the integrated CoS virtualization model, also showing the state-of-the-art on resource allocation and task scheduling in CoS. Chapter 3 discusses virtualization in CoS, and describes our proposed CoS virtualization model, Olympus. Chapter 4 describes our approach to the resource allocation in CoS, presenting Zeus. Chapter 5 describes our experiments and results regarding the evaluation on Zeus. Chapter 6 discusses related work. Chapter 7 portrays the concluding remarks and discusses future research directions.

2 CoS architecture and the integrated virtualization model

In this Chapter, we first detail the three-tier CoS architecture considered in this thesis (Section 2.1), and then describe the functions of a CoS virtualization model (Section 2.2). Next, we review the state-of-the-art on solutions for task scheduling and resource allocation (Sections 2.4 and 2.5, respectivelly), highlighting how each solution approaches the challenges involved in such activities. Before discussing these existing proposals, in Section 2.3 we describe the criteria to be used throughout the text to organize the solutions that will be presented. As task scheduling and resource allocation share several characteristics and objectives, the same criteria are used to analyze the proposals for both activities. We conclude with Section 2.6, in which we present a summary of the results of our review from the state-of-the-art in task scheduling and resource allocation applied to CoS.

2.1 The CoS architecture

In this work, we consider the CoS architecture shown in Figure 1. Three tiers compose this architecture: sensors tier (ST), edge tier (ET) and cloud tier (CT).

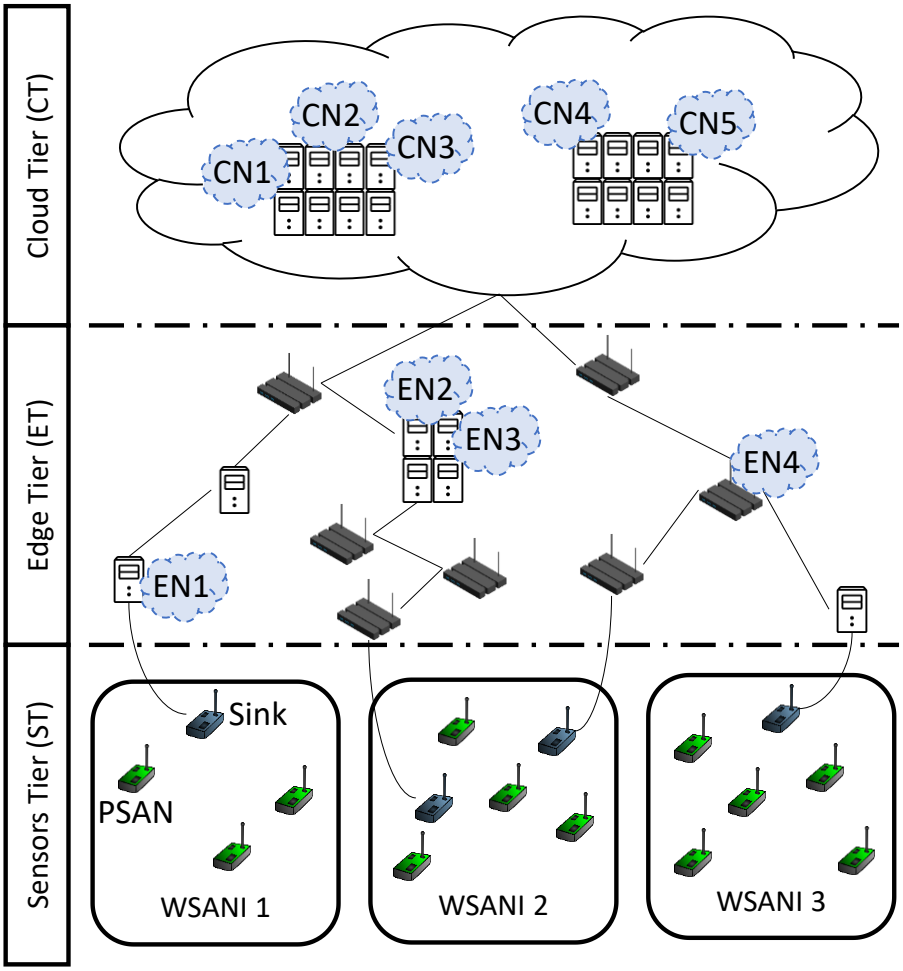

Figure 1. The three-tier CoS architecture

The sensors tier comprises the physical wireless sensor and actuator network infrastructures (WSANIs), each of which is owned and administered by an infrastructure provider (InP). Each WSANI comprises a set of physical sensor and actuator nodes (PSANs) deployed over a geographical area and connected by wireless links, so that every PSAN pertains to a single WSANI. Per Equation (1), we describe each PSAN in terms of processing speed PS, total memory TM, list of sensing units LS, list of actuation units LA, remaining energy EN, identification of the WSANI to which it pertains WI and information about its location coordinates LC.

$$psan_i = \langle PS, TM, LS, LA, EN, WI, LC \rangle \qquad (1)$$

In addition, we define that S_PSAN, in Equation (2), is the set of α PSANs existing in the sensors tier.

$$S_{PSAN} = \{ psan_i \mid i \in \mathbb{N}_0 \; and \; i < \alpha \} \qquad (2)$$

The InPs define the physical and administrative (logical) boundaries of their respective WSANIs. In our architecture, we assume that the physical boundaries of WSANIs (defined by the geographical area of deployment) may overlap, without influencing the administrative boundaries (defined by InP logic). Thus, the administrative boundaries may differ to the underlying physical boundaries, whose management is not in the scope of our work. Moreover, each PSAN has the knowledge of a valid communication path to reach the sink node within the WSANI. These communication paths are defined by underlying networking protocols chosen by the InP, such as [28][29][89][90][91][92] and are not in the scope of this thesis.

The edge tier comprises the edge nodes, which are typical physical edge devices. Such devices can be resource-poor devices such as access points, routers, switches, base stations, and smart sensors, or resource-rich micro-datacenters and machines, such as cloudlets [36]. Per Equation (3), we describe each edge node in terms of processing speed PS, total memory TM, bandwidth BW and its physical host identification HI.

$$en_i = \langle PS, TM, BW, HI \rangle \quad (3)$$

In addition, we define that S_EN, in Equation (4), is the set of β edge nodes existing in the edge tier.

$$S\_EN = \{en_i \mid i \in \mathbb{N}_0 \; and \; i < \beta\} \quad (4)$$

The cloud tier comprises the cloud nodes, which are multiple physical data centers (more powerful than physical edge devices) responsible for a global view of the CoS system. Cloud physical data centers are able to perform data-intensive computation, time-based data analysis and permanent storage of huge amounts of valuable data. The InPs can combine multiple physical data centers for providing services in the global scale, and each physical data center has a heterogeneous cost for providing its services. According to Equation (5), we describe each cloud node in terms of processing speed PS, total memory TM, bandwidth BW and its physical host identification HI.

$$cn_i = \langle PS, TM, BW, HI \rangle \quad (5)$$

Finally, we define that S_CN, in Equation (6), is the set of γ cloud nodes existing in the cloud tier.

$$S\_CN = \{cn_i \mid i \in \mathbb{N}_0 \; and \; i < \gamma\} \quad (6)$$

In our CoS architecture, we consider the existence of three main entities, namely the PSANs at the sensors tier, the edge nodes at the edge tier and the cloud nodes at the cloud tier. In our work, we choose to model PSANs as being the physical devices of the sensors tier themselves, while we consider the edge and cloud nodes as being virtual instances hosted by the physical devices of edge and cloud tiers, respectively. This choice allows us to consider that the deployment of each edge and cloud node on their respective physical hosts is transparent to our CoS architecture, and is handled by typical cloud and edge computing virtualization models. In addition, each edge and cloud node has information about the physical device that hosts it, so that we can associate the physical location of an edge or cloud node to the location of its physical host. In Section 2.2 we discuss issues on CoS virtualization, which is performed over the CoS architecture considered in this Section.

2.2 CoS virtualization

The capabilities of the CoS architecture, described in Section 2.1, are provided to users and their applications through a CoS virtualization model. Several works such as [57][58][59][60][7] correlate the virtualization of physical devices in CoS to the resource allocation and task scheduling problems, suggesting that the design of solutions to these problems needs to be jointly investigated to enable the CoS virtualization itself. It is important to mention that by CoS virtualization, we mean the decoupling between the physical infrastructure of the WSAN and applications via an abstract representation of the former, with the main purpose of sharing the physical infrastructure of WSANs. In our point of view, a full CoS virtualization model comprises the instantiation of virtual nodes (VNs) and at least two sub processes, one for performing resource allocation and the other for task scheduling. Figure 2 summarizes the relation between both resource allocation and task scheduling in CoS.

In the CoS paradigm, end users, with any given level of application domain expertise, define and implement applications. An application is similar to a workflow that describes interactions among the services provided by VNs. In addition, we call each activity of the application workflow as a request, because its main goal is to request (demand) the services of VNs. Thus, each application consists of a set of requests (activities of workflows) that demand the resources of the CoS infrastructure. A request is a set of abstract commands

defined by users, which represent the application functional requirements, thus denoting an abstract service. In addition, application requests have non-functional requirements. Among them, the data freshness is one of the most relevant in a distributed system based on sensing data, such as the CoS [13]. This importance increases particularly in the context of systems composed of a set of autonomous data sources, where the data integration with different freshness values can lead to semantic problems, hindering the execution of applications. There are several definitions of data freshness in literature [13]. Among these definitions stands out the one that quantifies the freshness of a given data based on the time elapsed since its acquisition.

Applications typically access the CoS through the edge and/or cloud tiers. During the operation of the CoS, several applications access the CoS, probably simultaneously, posing their requests. To meet such requests, the CoS infrastructure must provide the outputs (data, in case of sensing, or controls, in case of actuation) as specified by the requests. Therefore, the CoS physical infrastructure has to perform tasks (generating a certain processing load on physical nodes) to provide such outputs with maximum data freshness. To avoid the re-execution of processing loads by the physical infrastructure, it is possible to meet other future requests by simply re-using outputs from previous executions, if they meet the data freshness requirement of the request. Moreover, these outputs could be stored at different tiers of the CoS architecture (sensors, edge and cloud tiers). Therefore, it is necessary to make a first decision to meet a given request in the CoS environment: *is it possible to meet the request using the data stored in the CoS environment (without engaging physical nodes), or is it necessary to dispatch tasks to run directly on the physical infrastructure*? In the latter case, a second decision follows, regarding *how to distribute the processing load among physical nodes that make up the CoS physical infrastructure*. The first decision directly relates to the resource allocation process. Since this is a data acquisition process (either from real or virtual nodes), in our work we investigate it under the prism of data provisioning [42][43]. The philosophy behind data provisioning consists of abstracting, from applications, data acquisition/access, and allowing data sharing among applications, while meeting respective application requirements. The second one is a task scheduling decision. Traditional task scheduling algorithms in typical devices of the CoS [24] are responsible for selecting a group of physical

nodes that are suitable for the execution, in a given order, of the various tasks necessary to meet an application request.

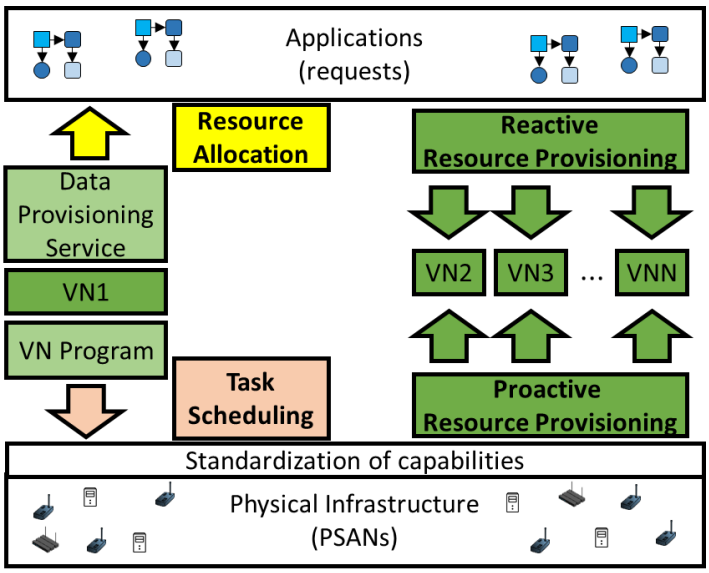

Figure 2. Virtualization, resource allocation and task scheduling in CoS

In line with the concept of WSAN virtualization [40], the CoS system comprises the several VNs created to represent the abstraction foreseen by data provisioning. Throughout this thesis we define the VN as a computational entity (a software instance). The VN has the main goal of abstracting to users not only the data, but also the computation and communication capabilities provided by a set of underlying nodes. Thus, as a single virtual entity, the VN simplifies the representation of its underlying infrastructure to users. The underlying nodes are abstracted as services that the VN provides to applications. In the CoS, such software services mediate the interaction between applications and physical entities. Services expose resources, defined as software components that provide data from or control the actuation on physical entities [44]. In addition, the VN has the secondary goal of coordinating the execution of tasks by the underlying nodes, required to perform its first main goal. Thus, the implementation, the scheduling and the execution of tasks in the physical infrastructure is the responsibility of VNs.

Therefore, we consider that every VN has a VN program, whose objective is to perform a series of tasks on the PSANs to update the data further provided by the VN through its data provisioning service. We consider that a VN program does not change during the existence of the VN. Thus, in CoS, task scheduling refers to the process, performed by the VN, of scheduling

tasks of each VN program to a given set of nodes from the underlying physical infrastructure of the CoS. We consider the existence of a service at each VN that is responsible for running the task scheduling process. Moreover, we consider another service at each VN that implements the process of execution and supervision of the execution of the tasks, scheduled by the former process, on the underlying physical infrastructure. Similarly, a VN has a process for performing resource allocation and a process to execute applications requests on VNs allocated by the former process.

Finally, a process to instantiate the virtual nodes is first necessary, in order to select and prepare the underlying physical infrastructure for use. In the context of CoS, resource provisioning refers to this process of instantiating virtual nodes. The term “provisioning” often refers to the action of equipping or preparation of an infrastructure for some purpose. As in traditional cloud computing systems, the process of resource provisioning [94] is responsible for managing the association of physical computational capacities to counterpart virtual entities. This association, in the CoS, must maximize the utilization of physical computational (and data) capacities to virtual nodes, while respecting the physical computational (and data) capacities constraints of the physical CoS infrastructure. Based on [94], proactive Resource Provisioning denotes the cases where the instantiation of a VN occurs before the allocation of the VN to an application request. VNs are instantiated by the infrastructure providers at a time prior to the execution of the CoS infrastructure and the arrival of applications. In case of reactive Resource Provisioning, the instantiation of a VN may also occur (in addition to the reuse of existing instances) in response to the need of its allocation to an application request (i.e. the requirements of a new VN are tailored to meet a specific allocation). Thus, VNs are instantiated on demand, by a dynamic (real-time) and automated resource provisioning algorithm. Generally, resource provisioning and adjustment according to demand should occur dynamically, in an elastic and transparent way.

It is important to mention that in our previous work [12] we concluded that the best theoretical position for positioning VNs is the edge tier. Among PSANs, edge nodes and cloud nodes, we chose the edge nodes to run VNs for the following reasons. The edge nodes have greater computation and communication capabilities than PSANs, and thus are able to manage VNs instances more efficiently. Moreover, edge nodes are in the edge of the network, closer to WSANIs and to the end users than the cloud nodes. Thus, edge nodes are in a

privileged position, in relation to the cloud nodes, for linking PSANs from different WSANIs under the same VN, and for reducing the latency for time sensitive applications. In addition, the cloud nodes store the registries of existing VNs in the whole CoS. Thus, we consider that edge nodes are the possible hosts for the VNs instantiated by the resource provisioning process.

We formally define the VNs and the set of VNs as follows. Per Equation (7), we describe each VN in terms of its list of underlying PSANs LU, services description SE and its host EN identification HI.

$$vn_i = \langle LU, SE, HI \rangle \quad (7)$$

Per Equation (8), we define a set S_VN containing all the θ VN instantiated by Infrastructure Providers (InPs).

$$S\_VN = \{vn_i \mid i \in \mathbb{N}_0 \; and \; i < \theta\} \quad (8)$$

Finally, designing solutions to the processes of task scheduling and resource allocation for CoS is still a challenge. The concepts of resource allocation and task scheduling are not new in the literature. Several areas of research closely correlated to CoS, such as IoT and WSANs, study these concepts with slight differences of context. Such differences relate mainly to the nature of the resources to be scheduled or allocated. In the following sections, we chose, mainly, to depict the works that pertain to our area of research, the CoS. However, since the area of CoS is recent, only few studies exist proposing task scheduling and resource allocation solutions. Therefore, we will discuss task scheduling and resource allocation solutions under a broader perspective, including the areas of IoT and WSAN. We choose the most recent solutions from other areas than the CoS that require little or no adaptation when applied to operate in the CoS.

It is important to mention that several resource allocation and task scheduling solutions exist for the area of cloud computing [68], which is also closely related to CoS. Such solutions ignore the specificities of the CoS environment. In traditional cloud computing, the physical infrastructures contain several data centers with high computing capacity servers, where the most important is simply to provision their processing capabilities, storage and communication data. When operating in a CoS environment, the need for provisioning sensing and actuation, the heterogeneity and the computational resource constraint nature of devices

pose new challenges to perform the resource provisioning. However, the area of resource allocation and task scheduling for cloud computing [41][48] has already been extensively surveyed, and thus we excluded this area from our review of the state-of-the-art.

Finally, based on the concepts of resource allocation and task scheduling presented in this Section, and that the best theoretical position for positioning VNs is the edge tier [12], our first macro criterion to classify proposals is to divide them into “proposals for task scheduling that can be applied at the sensors tier” and “proposals for resource allocation that can be applied at cloud and edge tiers”. Within these two macro categories, we identify and classify the existing proposals based on a set of additional criteria, described in the following Section.

2.3 Classification criteria

As a **first criterion** for classifying proposals for task scheduling and resource allocation, we consider the decentralization degree of solutions. In general, although centralized solutions are simpler to implement and have a global view of the network, they are well known as being more susceptible to failures and less scalable. In centralized task scheduling algorithms, a single VN placed at the sink node in the WSAN or a manager node in the cloud, is responsible for deciding how, when and where to perform the tasks. In turn, in decentralized task scheduling algorithms, multiple PSANs, known as the local schedulers, determine the scheduling. The local schedulers do not need to perform scheduling for all nodes, because different nodes may have different areas of interest, contributing to the scalability of the solution. Kapoor et al. [95] show that, among the several centralized and decentralized algorithms they proposed, the performance of the best-centralized algorithm is comparable to that of the best-decentralized algorithm for smaller systems. For medium and high system loads, the decentralized algorithms demonstrate a significantly higher performance in comparison to the centralized algorithms. Moreover, a hybrid approach can be adopted (partially decentralized), sharing characteristics of both centralized or decentralized solutions, sometimes combining two decision phases, one centralized and the other decentralized.

As a **second criterion** to classify existing solutions, we consider the ability of considering priorities during the task scheduling or the resource allocation process. Priorities can be considered among different application requests, tasks of VN programs, VNs, or PSANs. In the context of resource allocation, there may be applications requests with higher priority, concerning their degree of criticality (response time). For instance, a HVAC application may be

less critical than a fire detection application. This priority can also concern the amount of resources provided to the applications (VNs and PSANs), compared with other applications that are sharing the physical infrastructure. Regardless the kind of priority, such information regarding priority, possibly provided by users, could be used to rank tasks, requests, VNs, and the PSANs, which should be assigned first during task scheduling or resource allocation. For instance, in resource allocation, if a more priority application arrives, the VN must stop the request currently running and queue the requests of the highest priority application according to its priority.

As a **third criterion** to be used when analyzing and classifying existing proposals, we consider their ability to handle precedence relationships (dependencies between inputs and outputs) among requests and among tasks. Most works, such as [56], consider applications/VN programs that comprise a single request/task regarding data acquisition. This is a common approach, for instance, in task scheduling algorithms for traditional WSANs. In such algorithms, the VN programs have the interest on mere data acquisition (represented as a single data acquisition task), without considering the actuation, computation and communication capabilities of WSAN nodes. In traditional WSANs, the task scheduling algorithm decides only about which nodes will perform each single task, and at which time such execution will occur. Thus, since tasks have no precedence relationships, there is no need to model VN programs through, for instance, the usual representation of a Directed Acyclic Graph (DAG). However, in scenarios where VN programs consider multiple tasks (possibly mixing data acquisition, actuation, computation and communication tasks) with precedence relationships, it is very important to model such relationships, in order to make a proper decision. This applies to either task scheduling or resource allocation, because PSANs or VNs must perform first the requests or tasks whose inputs are satisfied, and are free to start being processed. It is important to mention that the representation of requests and tasks through a DAG, and handling the precedencies among them is a feature explored by few works. Moreover, it is one first step towards another feature not explored by any work so far, which is representing an application or VN program as a complex structure of standard information fusion procedures (each request or task as being an information fusion procedure).

We consider the ability of sharing the results of execution among requests or tasks as a **fourth criterion**, which is relevant to analyze the proposals and their benefits. Some

solutions for task scheduling share the results of tasks that are common among multiple VN programs. These proposals perform the tasks in common (i.e., tasks that will serve different VN programs) only once, sharing the results to further improve the resource utilization of PSANs.

As a **fifth criterion** for classifying proposals, we consider the adoption of what we call a *full device virtualization model* by these proposals. A full virtualization model is defined as a process to virtualize the data and computational/communication capabilities from PSANs through the instantiation of VNs, also comprising task scheduling and resource allocation procedures (mostly adopted by the proposals for the areas of CoS and IoT).

As a **sixth criterion**, we consider the ability of supporting time-based (pull) and event-based (push) applications (or VN programs) simultaneously. A time-based (TB) application is the one for which the application decides the exact moment in time for demanding the VN (or PSANs) resources. Thus, the pull model attends time-based applications (or VN programs). For instance, in the context of a pull model in resource allocation, a VN demands its underlying physical infrastructure and responds to the application at the time defined by the application. This is the most conventional type of application, and it is perfectly suited to the context of resource allocation due to the temporal aspect, i.e. VNs can have full control about the time duration of each request, and do not need to wait for events with random occurrence times in future (interrupts), to finish their processing. Thus, pull applications (VN programs) are supported by all resource allocation (task scheduling) proposals to be discussed in Sections 2.4 and 2.5. An event-based (EB) application (VN program) is the one that demands the VN (PSAN) resources only when it detects the occurrence of a specific event of interest by the application (VN program). Thus, a push model attends event-based applications (VN programs). For instance, in the context of a push model in resource allocation, the VN demands its resources at a moment in time that is unknown, previously to the occurrence of the event of interest by the application. The push model is a typical publish-subscribe model, in which applications subscribe their interests to a VN, which, in turn, waits until an event occurs to publish its results respective to this event. The ability to support both models simultaneously makes the approach for resource allocation or task scheduling more general, thus fitting well in a high-scale deployment of a CoS infrastructure, meeting a broader range of applications.

As a **seventh criterion**, we classify proposals according to the characteristics of the optimization problem formulated to meet the resource allocation and task scheduling objectives. There are several methods in the literature that can be used to mathematically formulate the resource allocation and task scheduling problems [17], [66]. In most cases, the optimization problem is referred as an integer program, in which the decision variable (with respect, for instance, to which PSAN will be allocated to which task) is a binary variable. Some other real-valued criteria may also be of use, and thus could be included in the problem formulation, such as when defining the start times of tasks. When including real-valued decision variables the optimization problem is referred as a linear program. Another commonly used option is to formulate the problem as a multi-objective optimization problem [66]. In contrast to the previous cases of single goal problems, the multiple goals conflict with each other, i.e., to maximize the attendance to application requirements results in an increased consumption of resources, and vice versa. That is, it is not possible to find a single solution that minimizes an objective and maximizes another simultaneously. In the multi-objective approach, it is obtained an optimal solution-set (Pareto-optimal solutions) with numerous solutions indifferent to each other, according to some pre-established criteria, leaving the analyst to decide which solution to use, according to its own established criteria.

As an **eighth criterion**, we classify proposals according to the characteristics of the algorithm/heuristic used to solve the formulated optimization problem. To solve optimization problems in order to seek the optimal solution, there are a number of methods in the literature [67][66]. One example is the linear programming and its variants [67]. However, when considering the typical approach of representing requests and tasks as DAGs, the optimization problem becomes more difficult as the size of the DAG increases. In fact, finding a schedule of minimal length for a given DAG is, in its general form, an NP-hard problem [63][17], i.e. problems whose explosive combinatorial nature hinder the quick search for optimal solutions when they grow [53]. That is, an optimal solution cannot be found in polynomial time (unless NP = P). Because of this kind of problem, an entire area emerged that deals with quick search of solutions, ranging from the theoretical analysis to heuristics and approximation techniques that produce near optimal solutions. Thus, the problems tackled in resource allocation and task scheduling fit the class of NP-hard problems, justifying the application of heuristic techniques.

Heuristic techniques approach the optimal solution, solving the problem by obtaining sub-optimal solutions in reduced computation time, once the search for the optimal solution is much more computationally intensive. Heuristic based proposals can follow, for instance, traditional graph theory, evolutionary, game/auction, greedy, machine learning, Voronoi diagrams or probabilistic approaches [68]. Evolutionary algorithms are typically used to provide good approximate solutions to problems that cannot be easily solved using other techniques. Due to its random nature, it is not guaranteed that evolutionary algorithms find an optimal solution to the problem, but they will often find a good solution, if any. Genetic algorithms have proven to be a successful way to produce satisfactory solutions to many problem formulations. Finally, Game theory is increasingly being used as a modeling and design framework in decentralized algorithms. A distributed game-theoretic approach to task allocation provides autonomy to sensor nodes, which can decide the best scheduling in actual neighboring context. In solutions following such approach, communications are made only between certain sensors in a neighborhood, which is a potentially energy efficient and scalable solution.

As a **ninth criterion**, we consider the presence of the edge tier in the proposed architectures [4]. In such proposals, the devices in the edge tier (edge nodes, sinks, gateways, for instance) play an active role in the resource allocation or task scheduling process, allowing solving the problem in a distributed way.

Finally, the **tenth criterion** regards considering physical devices (sensors / things) as active resource providers, providing to applications (VN programs) computing and actuation capabilities, and not being mere passive sensing data sources.

2.4 State-of-the-art on task scheduling at the sensors tier

In this Section, we describe relevant works found in literature that represent the state-of-the-art on task scheduling at the sensors tier of the CoS architecture. The described works present proposals either tailored for the specific field of CoS [105][4][106][107], or for the broader field of IoT [109][117][118][108][110], or for the more traditional field of WSAN [54][24][116][113][95][115][56][114][111][112][51].

Among the works that propose task scheduling for CoS, Zhu et al. [105] is an example of a centralized solution. In their work, the authors analyze the characteristics of task scheduling with respect to integrating cloud computing and WSANs, proposing two novel task

scheduling algorithms. Their algorithms follow a heuristic, greedy based approach, and are able to handle priorities among tasks. In their algorithms, the main goal is to divide tasks into two groups: tasks to be performed in the WSAN (G1) and tasks to be performed in cloud (G2). For all tasks submitted in group G1, with higher priority, the algorithm uses a set of rules to decide (based on the costs of using the resources available) for the best schedule. After, the same procedure is performed for group G2, with lower priority.

Phan et al. [4] is another example of a centralized solution. They proposed a cloud-integrated WSAN architecture, and studied the optimization of a push-pull communication scheme among the three layers of their architecture using a genetic algorithm. Therefore, they are among the proposals that can support both time-based and event-based applications, which guarantees they cover a broad spectrum of application domains. In addition, in this proposal the devices in the edge tier (edge nodes, sinks, gateways, for instance) play an active role in the task scheduling process. Moreover, their proposal includes a full device virtualization model. At the sensor layer, several heterogeneous WSANs embedded in the physical environment exist, using a tree topology. Nodes periodically read sensors and push data to the sink node. The edge layer is a collection of sink nodes, each of which participates in a certain sensor network and stores incoming sensor data in its memory, pushing them periodically to the cloud layer. Sink nodes maintain the mappings between physical sensors and virtual nodes. In addition, each sink receives a “pull” request from a virtual node when the virtual node does not have the data that an application requires. If the sink node has the requested data in its memory, it returns that data. Otherwise, it issues a pull request to a sensor node that is responsible for the requested data. The cloud layer operates on one or more clouds to host end-user applications and management services for the applications. Applications are operated on virtual machines in clouds, and always access physical sensors through virtual nodes. Users are assumed to place continuous sensor data queries on virtual nodes via cloud applications in order to monitor the physical environment. If a virtual node already has data that an application queries, it returns that data. If a query does not match, the virtual node issues a pull request and sends it to a sink node. Phan et al. focus on two services in their virtualization model. The first is the sensor manager, which virtualizes physical heterogeneous sensors in a unified way by abstracting away their low level operational details. The second is the communication manager, responsible for push-pull hybrid communication

between different layers. The key component in the communication manager is the communication optimizer, which solves an optimization problem to seek the optimal data transmission rates for sensor and sink nodes with respect to multiple optimization objectives (maximize sensor data yield for applications, minimize bandwidth consumption between the cloud and edge layers and minimize energy consumption in the sensors layer).

Dalvi [106] proposed a centralized task scheduling scheme to minimize energy consumption in CoS, which is based on TDMA, spatial correlation and on the Voronoi diagram. The Voronoi diagram is used for representing WSANs in a CoS. In the diagram, a rectangular region represents the area from which the user needs data. The area covered by the WSAN is divided into small cells that are centered at points. Each point in the cell represents a wireless sensor in a WSAN. The edges of each cell are formed by connecting perpendicular bisectors of the segments joining all neighboring points. The sensor located at the center of a cell can sense data for the area covered by cell. This data is more accurate as compared to the data sensed by other sensors for that region. This Voronoi diagram is built when WSN is initialized. A calculation is performed to find the minimum number of nodes required to cover the area selected by the user. The allocation scheme is based on the concept that sensors in densely deployed zones will have more number of neighbors compared to sparsely deployed zones and hence more number of edges in Voronoi diagram. As the previously discussed work [4], the solution presented in [106] also supports both pull and push communication models, and includes a full device virtualization model, similar to the one in [4]. In the virtualization model proposed by Dalvi, there are three layers: client centric, middleware, and sensor centric. The client centric layer connects end users to the CoS, managing a user's membership, session, and GUI. Middleware is the heart of the CoS, managing virtual nodes with help from components such as provision management, image life-cycle management, and billing management. The sensor centric layer connects the middleware to the physical WSNs. It also registers participating WSNs, maintains participating WSNs, and collects data.

Yao et al. [107] proposed a centralized and adaptive task scheduling mechanism to schedule optimally the transmission opportunities of devices, considering multimedia distortion reduction, hidden node problem, transmission interference, and signal coverage. Their proposed mechanism adopts a heuristic-based, greedy approach for performing the task scheduling.

Among the works that propose task scheduling solutions for IoT, Kim & Ko [117] present a centralized task scheduling approach based on genetic algorithm to minimize the amount of data transmissions between mobile devices in IoT. They transformed the task scheduling problem into a variant of the degree-constrained minimum spanning tree problem and applied a genetic algorithm to reduce the time needed to produce a near-optimal solution.

Li et al. [109], Li et al. [108] and Billet et al. [110] are also examples of centralized solutions for task scheduling in IoT. Li et al. [109] proposed a genetic algorithm based on a teaching and learning technique for scheduling tasks with multiple restrictions of relations in processing sequences in IoT. In their genetic algorithm specification, the crossover operator is used in the swarm intelligent algorithms to learn information from other solutions, thus converging to the optimal search space faster. Li et al. [108] proposed a QoS scheduling model for IoT, which explores optimal QoS-aware services composition at application layer, heterogeneous network environment at network layer, and the information acquisition for different services at sensing layer. Billet et al. [110] proposed a binary programming problem formulation for task scheduling for IoT, along with an efficient heuristic for solving it, based on location, capabilities and QoS constraints.

Among the works that propose task scheduling solutions for WSANs, Farias et al. [54] proposed a framework for Shared Sensor and Actuator Networks (SSAN), including an energy-efficient centralized task scheduling algorithm. A major feature of their work is that the algorithm performs tasks in common to multiple applications only once. In other research, Li et al. [24], introduce a task scheduling algorithm exploiting the fact that different applications may share the same sensing data with common QoS requirements, as well as spatial and temporal characteristics. Both proposals promote the cost-effective utilization of the resources available in the shared infrastructure, aiming to increase the ROI (return of the investment) for the owners. They support different priorities and precedence relationships among tasks. Furthermore, both support time-based (pull) and event-based (push) applications simultaneously. Also in the context of SSANs, Bhattacharya et al. [112] proposed an integrated application deployment system that performs task scheduling based on their Quality of Monitoring (QoM) of physical phenomena due to the close coupling of the cyber and physical aspects of distributed sensing applications. Therefore, the task scheduling

algorithm deals with the inter-node Quality of Monitoring (QoM) dependencies, typical in cyber-physical applications. The QoM of a distributed sensing application usually depends on the set of nodes allocated to it. Moreover, the measurements of different sensors are often highly correlated resulting in inter-node dependency, i.e., the QoM contributed by a node to an application is dependent on the other nodes allocated to the same application. Since the SSAN paradigm aims at fully exploiting the deployed sensing infrastructure for executing multiple applications with distinct requirements, it is a desirable feature that all possible communication patterns are support. Therefore, the work described in [112] schedules both time-based and event-based applications. However, the authors do not mention the handling of different priorities, which is another desirable feature for SSNs.

Our research group, in [54] and [24], proposed centralized algorithms for scheduling tasks in WSANs. These algorithms, in addition to being concerned about saving energy by choosing the best node for a particular task, also perform tasks in common for different applications only once, sharing the results of these applications to improve the use of the limited resources of physical nodes. This sharing takes advantage of the fact that the same physical infrastructure is used by multiple applications, as is the case in virtualized environments. These tasks in common are identified by preprocessing the DAGs of applications, before starting the task scheduling process. These two works also include a full device virtualization model, in which the sensors play an important role as service providers. In their virtualization model, three major elements exist, namely web server, sink nodes and sensor nodes, organized in a hierarchical manner. The web server acts as a frontier to handle arrivals of applications and performing task scheduling. Final users, through applications, request different services from the system and the web server acts as a service provider to reply those requests. After receiving requests from the web server, the sink node of each WSN schedules these tasks to individual sensor nodes. Sensor nodes send the descriptions of their services to sink nodes, which keep them in a repository.

Dai et al. [116] propose a multi-objective algorithm for centralized task scheduling in WSANs, seeking to optimize the total make span of tasks, but meanwhile, also paying attention to the probability of node failure (related to unsuccessful task performing) and the lifetime of network.

Hu et al. [114] propose three different greedy-based centralized algorithms to optimize data fusion parameters in the WSAN task scheduling problem, modeled as an Integer Linear programming problem. Their proposal includes a full device virtualization model. In their model, a given virtual sensor is instantiated and cannot change during a given time slot. They assume a common sensing period, with each sensor generating samples and making a local decision as to whether an interesting event occurred. Event arrival times are independent and their distribution is known in advance. The events of interest are, in theory, detectable by the available physical sensors. Their virtualization model comprises two pre-deployment independent functions: a function for forming information fusion rules, and a function for assigning optimal virtual sensors and their scheduling. Fusion rules are formulated through either synthetic or experimental data. The intuition is that the fusion rules define the optimal set of sensors for each event as well as the algorithm to combine the sensor readings.

Rowaihy et al. [56] proposed centralized and decentralized energy-aware solutions to the problem of optimally scheduling multiple missions to WSANs, in which each mission uses its specific and exclusive subset of sensor nodes. The problem of mission-sensor scheduling is modeled as a weighted bipartite graph to optimally schedule the sensors for missions. Their proposed solutions build on traditional graph theory and are able to handle priorities among missions, relating priority with mission profit, i.e. these solutions schedule missions that have higher profit first than other missions (missions are sorted in order of decreasing profit). In the proposed weighted bipartite graph, the vertex sets consist of sensors and tasks. A positively weighted edge means that a sensor is applicable to a task. The weight of the edge indicates the utility that the sensor could contribute to the task. The authors seek a semi-matching of sensors to tasks, so that (ideally) each task is satisfied. Their proposed solutions perform graph manipulation to represent different problem variants, with different constraints. Moreover, they propose several greedy algorithms, jointly with the graph theory approach. One of their greedy algorithms considers tasks in decreasing order of profit. For each task, the algorithm assigns available sensors in decreasing order of offered utility, until the mission is satisfied.

Li et al. [51] and Edalat et al. [111] are examples of hybrid approaches, meaning that they inherit features of both centralized and decentralized solutions. In Li et al. [51] the authors proposed a heuristic-based three-phase algorithm for allocating tasks to multiple

clusters in hierarchical WSANs, seeking to minimize the overall energy consumption and balancing the workload of the system while meeting the applications deadlines. The task scheduling problem is referred as an integer program and solved as a multi-objective problem. The proposal considers tasks with different priorities and dependencies among the data input and outputs of tasks, represented through DAGs. Moreover, the sensors are considered as active resource providers, instead of being mere passive data sources. Edalat et al. [111] proposed a heuristic two-phase winner determination protocol to solve the task scheduling problem modeled as a distributed reverse combinatorial auction, seeking to maximize WSAN lifetime while enhancing the overall application QoS, in terms of deadlines. They also consider tasks with different priorities and dependencies among the data input and outputs of tasks, represented through DAGs.

In terms of distributed solutions, Wu et al. [113], present a distributed game-theoretic approach for task scheduling in SSANs based on the correlation among sensor readings from different nodes. Their approach supports time-based and event-based communications.

TABLE 1. CLASSIFICATION OF TASK SCHEDULING PROPOSALS

	1 Degree of decentralization			2 Handles priorities	3 Handles precedencies	4 Shares requests/tasks in common	5 Full device virtualization model	6 Type of application		7 Optimization problem formulation				8 Approach of the algorithm proposed/used to solve the problem								9 Considers the edge tier	10 Devices as active resource providers
	1.1 Centralized	1.2 Hybrid	1.3 Decentralized					6.1 Time based	6.2 Event based	7.1 Integer	7.2 Linear	7.3 Non-linear	7.4 Multi-objective	8.1 Graph theory	8.2 Evolutionary	8.3 Game / auction	8.4 Greedy	8.5 Machine learning	8.6 Voronoi Diagram	8.7 Lin. programming	8.8 Probabilistic		
[112]	X							X	X	X							X						
[54]	X				X	X	X	X	X					X			X						
[24]	X			X	X	X	X	X	X				X	X			X						X
[108]	X			X			X	X			X						X						
[109]	X				X			X							X			X					
[110]	X				X			X	X	X	X						X						
[107]	X							X									X						
[105]	X			X				X									X						
[114]	X						X	X		X							X						
[4]	X						X	X	X				X		X							X	
[116]	X							X					X		X								
[106]	X						X	X	X										X				
[117]	X							X						X	X								
[111]		X		X	X	X		X	X	X						X							
[51]		X		X	X			X		X	X		X	X			X						X
[56]			X	X				X		X	X			X			X						
[113]			X					X	X	X						X							
[95]			X					X									X						
[115]			X	X	X			X									X						
[118]			X	X				X	X							X							

Wang et al. [115], propose a cluster-based task scheduling algorithm based on a greedy approach that balances the node energy consumptions, where the sensor nodes are classified into different ranking domains, and that supports requirements of real-time, heterogeneity, flexibility and scalability. Different priorities among application tasks are supported and the precedence relationships of tasks are considered in their scheduling decisions.

Kim [118] proposed a task scheduling solution to optimize bandwidth in IoT based on a cooperative game and on the concept of Shapley value. They adopt a fully decentralized approach and leverage the collaboration among the devices to achieve the application goals. Their approach is able to handle tasks with different priorities.

Finally, besides comparing proposals for task scheduling regarding their decentralization degree, Kapoor et al. [95] proposed their own greedy-based decentralized task scheduling algorithms. They proposed the Random Allocation algorithm, which does not use any information about the application or the network while making a scheduling decision. They proposed the CPU Load Balanced Allocation and Data Load Balanced Allocation algorithms, which schedule tasks making decisions to balance the CPU usage and data transmissions, respectively, among nodes. They also proposed the Balanced Metric Allocation

algorithm, which aims to balance the energy consumption, both due to the CPU component and the radio component amongst all the sensor nodes. Finally, they also proposed the Maximum Energy First algorithm, which selects first, from the set of available nodes, the sensor nodes that have the highest available energy, for execution of the tasks.

Table 1 presents a summary with the task scheduling proposals classified according to the 10 criteria considered in this Chapter. In the following section, we present the state-of-the-art on resource allocation at edge and cloud tiers, and we classify the respective proposals according to the criteria used in this section.

2.5 State-of-the-art on resource allocation at edge and cloud tiers

In this Section, we describe relevant works found in literature that represent the state-of-the-art on resource allocation at the edge and cloud tiers of the CoS architecture. The described proposals were developed either for the field of CoS [119][46][120][49] or IoT [71][70][73][121][122][75][123][124]. We will analyze the proposals in the light of the same criteria used for task scheduling solutions. Almost all works, with a single exception, propose centralized solutions for the resource allocation problem.

Among the works that propose resource allocation in CoS, Delgado et al. [46][119] proposed an heuristic algorithm and optimization framework to perform resource allocation, seeking to maximize the number of applications sharing the CoS, while accounting for the limited storage, processing power, bandwidth, and energy consumption requirements of sensors tier. Their resource allocation algorithm follows a traditional linear programing technique.

Misra et al. [120] present a mathematical formulation of CoS, with a thorough evaluation of the cost effectiveness of CoS by examining the costs of sensor nodes due to deployment, maintenance, and rent by users, as well as the profits in terms of the service acquired from the sensed data, always from the perspective of every user of the CoS. In the full device virtualization model used by Misra et al., there are three-tiers (users/applications, virtual nodes and sensors). The communication interface of a user is primarily a Web interface running at the site of the cloud service provider. It is a Web portal, through which the user requests the CoS. The user is kept abstracted from the underlying complex processing logic required due to perform resource allocation, application-specific aggregation, and virtualization. Moreover, sensor nodes are heterogeneous, thus the sensor nodes are

standardized using a Sensor Modeling Language. Every physical sensor node reports its sensed data to the CoS storage. Within the cloud environment, the sensed data are aggregated in real time.

Dinh et al. [49] propose an interactive model for the CoS to provide on-demand sensing services for multiple applications with different requirements, designed for both the cloud and sensor nodes to optimize the resource consumption of physical sensors, as well as the bandwidth consumption of sensing traffic. Dinh et al. consider requests in common in their solution. Their approach formulates a unique DAG of requests, merging requests in common for multiple applications. In their approach, requests in common are merged by considering the more restrictive application requirements.

Among the works that propose resource allocation in IoT, Narman et al. [71] propose a dedicated server allocation for heterogeneous and homogeneous systems, to provide efficiently the desired services by considering priorities of applications requests. Yu et al. [70] proposed a cloud-based vehicular network managed by a strategy based on game theory to optimally allocate resources, together with virtual machine migration. Both proposals include a full device virtualization model. The first work [71] also includes mechanisms to handle priorities among application requests, using this priority value to decide the next request to be served and allocate the amount of resources for each request.

Angelakis et al. [121] presented a mathematical formulation of assigning services to interfaces with heterogeneous resources in one or more rounds, and developed two algorithms to approximate the optimal solution for big instance sizes.

Zeng et al. [73] proposed an edge computing supported software-defined embedded system, together with the formulation of a resource allocation problem as a mixed-integer nonlinear programming problem, and a computation-efficient solution. Vögler et al. [122] propose an infrastructure that provides elastic provisioning of application components on resource-constrained and heterogeneous edge devices in large-scale IoT deployments, which supports push-based as well as pull-based deployments. Moreover, their infrastructure also manages time precedence restrictions among application tasks. Aazam et al. [75], in their proposed methodology for resource estimation and management through edge computing, formulate resource management based on the fluctuating relinquish probability of the customer, service type, service price, and variance of the relinquish probability. In their paper,

they extended their previous model to include a customer probabilistic resource estimation model, to manage the resources for IoT devices. Different priorities among application tasks are considered in their solution. Abedin et al. [124] provide an efficient IoT node pairing scheme between the same domains of IoT nodes in edge paradigm, based on the Irving's matching algorithm, and model the problem as a one sided stable matching game with quota. All these works ([73][122][75][124]) consider the presence of the edge tier in their proposed architectures.

Aazam et al. [123] present a service oriented resource management model for IoT devices, using edge computing. Their work is mainly focused on considering different types customer and device based resource estimation and pricing, even in presence of mobility. In their proposed model, sensors, IoT nodes, devices, and cloud service customers (CSCs) contact the edge to acquire the required service(s) at best price. CSCs perform the negotiation and service level agreement (SLA) tasks with the edge. The edge is in charge of estimating the consumption of resources, so that they can be allocated in advance.

Only the work described in [70] proposed a decentralized approach to resource allocation. The authors proposed to integrate cloud computing into vehicular networks such that the vehicles can share computation resources, storage resources, and bandwidth resources. Their proposal includes a full device virtualization model, including a vehicular cloud, a roadside cloud, and a central cloud. The authors study resource allocation and virtual machine migration for effective resource management in this cloud-based vehicular network. A game-theoretical approach is presented to optimally allocate cloud resources. Virtual machine migration due to vehicle mobility is solved based on a resource reservation scheme.

TABLE 2. CLASSIFICATION OF RESOURCE ALLOCATION PROPOSALS

	1 Degree of decentralization			2 Handles priorities	3 Handles precedencies	4 Shares requests/tasks in common	5 Full device virtualization model	6 Type of application		7 Optimization problem formulation				8 Approach of the algorithm proposed/used to solve the problem								9 Considers the edge tier	10 Devices as active resource providers
	1.1 Centralized	1.2 Hybrid	1.3 Decentralized					6.1 Time based	6.2 Event based	7.1 Integer	7.2 Linear	7.3 Non-linear	7.4 Multi-objective	8.1 Graph theory	8.2 Evolutionary	8.3 Game / auction	8.4 Greedy	8.5 Machine learning	8.6 Voronoi Diagram	8.7 Lin. programming	8.8 Probabilistic		
[71]	X			X			X	X									X						
[121]	X						X	X		X	X						X						X
[119]	X							X		X	X									X			
[46]	X							X		X	X									X			
[120]	X						X	X															
[122]	X				X		X	X	X													X	X
[73]	X							X		X		X								X		X	
[75]	X			X				X													X	X	X
[123]	X			X				X													X		X
[124]	X							X								X						X	X
[49]	X					X	X	X			X									X			
[70]			X				X	X								X							

Table 2 presents a summary with the resource allocation proposals classified according to the 10 criteria considered in this Section. In Section 2.6, we present a summary of both the state-of-the-art on resource allocation and task scheduling. We present a macro classification of all works reviewed in this Section, under the criteria described in Section 2.3, and highlight the open issues in the state-of-the-art in task scheduling and resource allocation applied to CoS based on this classification.

2.6 Open issues

In this Section, we present a summary of the results of our review from the state-of-the-art on task scheduling and resource allocation applied to CoS. We categorized all the relevant proposals found in literature by their key differentials. We present this categorization in Table 3.

TABLE 3. CRITERIA FOR CLASSIFYING PROPOSALS

Criteria		Works on task scheduling	Works on resource allocation
1 Degree of decentralization	1.1 Centralized	[112][54][24][108][109] [110][107][105][114] [4][116][106][117]	[71][121][119][46] [120][122][73][75] [123][124][49]
	1.2 Hybrid	[111][51]	
	1.3 Decentralized	[56][113][95][115][118]	[70]
2 Handles priorities		[51][56][115][105][108] [111][24][118]	[71][123][75]
3 Handles precedencies		[111][51][54][24][109] [110][115]	[122]
4 Shares requests/tasks in common		[111][54][24]	[49]
5 Full device virtualization model		[54][24][108][114][4] [106]	[71][121][119][46][120] [122][70][49]
6 Type of application	6.1 Time based	[111][112][51][54][24] [56][113][108][109] [110][107][105][114][4] [116][106][117][95] [115][118]	[71][121][119][46][120] [122][73][75][123][124] [49]
	6.2 Event based	[111][112][54][24][113] [110][4][106][118]	[122]
7 Optimization problem formulation	7.1 Integer	[111][112][51][56][113] [110][114]	[121][119][46][73]
	7.2 Linear	[51][56][108][110]	[121][119][46][49]
	7.3 Non-linear		[73]
	7.4 Multi-objective	[51][24][4][116]	
8 Approach of the algorithm proposed/used to solve the problem	8.1 Graph theory	[51][54][24][56][117]	
	8.2 Evolutionary	[109][4][116][117]	
	8.3 Game / auction	[111][113][118]	[124][70]
	8.4 Greedy	[112][51][54][24][56] [108][110][107][105] [114][95][115]	[71][121]
	8.5 Machine learning	[109]	
	8.6 Voronoi Diagram	[106]	
	8.7 Lin. programming		[119][46][73][49]
	8.8 Probabilistic		[75][123]
9 Considers the edge tier		[4]	[122][73][75][124]
10 Devices as active resource providers		[51][24]	[121][122][75][123][124]

Regarding task scheduling, we can notice some open issues in existing works. First, despite the inherently distributed nature of CoS, IoT and WSN, more than half of the proposals for task scheduling are centralized. Although decentralized scheduling algorithms have been proposed, they still lack an important feature that is to promote the sharing of common tasks among the VN programs. We believe this is a key requirement to achieve an efficient solution mainly in large-scale deployments. In addition, besides sharing the tasks of VN programs, we claim that efficient and suitable solutions for CoS must take into account the priorities and precedencies among tasks, representing tasks and their precedencies through a DAG. Other relevant issue that is still poorly exploited in existing works is the proposal of a full device virtualization model, considering devices as active resource providers, other than passive data sources.

Regarding proposals for resource allocation, only one work found in the current literature proposed a decentralized solution, while all the others are centralized approaches. Moreover, only one proposal [49] was found on resource allocation for CoS that shares the

results of tasks in common among multiple applications. We identify only one proposal that considers time precedence restrictions among applications tasks [122], which is also the only one supporting time-based (pull) and event-based (push) applications simultaneously [122]. Three proposals [71][123][75] manage priorities among applications tasks, so this is also an underexploited feature in resource allocation. We believe this issue requires further investigations, since in a CoS scenario of large dimension and serving multiple applications, it is crucial to assure that more critical applications receive their required resources with some priority. As expected, a fair number of proposals consider the edge tier in their architectures [122][73][75][124]. Based on all the benefits such tier can bring, we believe this will be a trend, mainly for time critical applications and for scenarios with mobile devices. Finally, five proposals ([121][122][75][123][124]) consider the physical devices as actively engaging as the resource providers, instead of being mere passive data sources. We claim that fully utilizing the resources provided by the sensors tier is the best strategy to build cost-effective CoS systems. However, this feature is still poorly exploited in the proposals.

Regarding all task scheduling and resource allocation proposals, it is important to mention that most solutions are either fully static, or handle partially the dynamic characteristic of the CoS environment when running the scheduling decision in cycles. In such partially dynamic approaches, the state of the system (acquired at the beginning of each cycle and used for making the scheduling/allocation decision) is assumed to remain the same during the entire duration of that cycle. This is not adequate when considering applications that handle continuous data streams [110]. To support this kind of application, it is necessary to model the state of each resource as a function that varies in time, thus fully handling the dynamic characteristic of the CoS environment, as in [110].

2.7 Conclusion

In this chapter, we described the works from the state-of-the-art on resource allocation and task scheduling in CoS. By classifying such works, we raised several open issues in the current literature not fully addressed by any of the proposals. Thus, this chapter contributes by showing the deepening in the investigation of research question 1 of this thesis, being the first result achieved. From this result, we selected the following open issues to address in this thesis, related with the investigation of research questions 2 and 3.

Regarding research question 2, respective to the proposal of a CoS virtualization model, we identify that there is a lack of a decentralized full device virtualization model, with devices playing an active role as resource providers and considering the edge tier in its architecture. In addition, this virtualization model should support effective, fast and lightweight algorithms to perform resource allocation and task scheduling. Finally, this virtualization model should present an application model which is able to model priorities and precedencies among requests and tasks, and the sharing of requests and tasks in common among multiple applications and VN programs. This virtualization model should also support both time-based (pull) and event-based (push) applications simultaneously.

It is important to mention that, in this thesis, we do not present an approach to task scheduling, due to the restriction of time to propose such a complex solution. However this is still an open issue that we suggest investigating, as future work. We suggest investigating fully decentralized proposals of task scheduling algorithms, such as the ones based on game theory [23][24][54]. In this thesis, we consider that such kind of algorithm is used to obtain data from within WSANs. In this thesis, we emphasize a deeper investigation on the proposal of a resource allocation algorithm, respective to research question 3.

Regarding research question 3, respective to the proposal of an algorithm to perform resource allocation in CoS, we identified the lack of formulations of the problem of resource allocation as a mixed integer non-linear problem, considering the maximization of the freshness of the data provided to applications. We also identified the lack of hybrid and heuristic algorithms, which are fast and lightweight, to find near optimal solutions to the problem of resource allocation. Moreover, a resource allocation algorithm should handle precedencies among requests and tasks, and the sharing of requests and tasks in common among multiple applications. We consider that such an algorithm must support at least time-based applications and a proactive resource provisioning processes.

It is important to mention that the following open issues in resource allocation will not be addressed by our proposal for a resource allocation algorithm in this thesis, and thus should be investigated as future work: (i) supporting both event-based and time-based applications simultaneously, (ii) supporting both proactive and reactive resource provisioning processes, (iii) handling priorities among applications.

In Chapter 3, we investigate research question 2, and present an overview of our own approach of a full device virtualization model, to support both resource allocation and task scheduling in CoS. This approach is called Olympus [40]. In Chapter 4, we investigate research question 3, and present our specific algorithm to perform resource allocation in CoS, called Zeus.

3 Virtualization in the Cloud of Sensors

In this thesis, our research started by studying CoS virtualization models and proposing our own approach, called Olympus [40]. The virtual nodes (VNs) provided by Olympus are built as an overlay on top of the three-tier CoS infrastructure described in Section 2.1. In the three-tier CoS infrastructure, both edge and cloud are typically virtualized, thus edge nodes (ENs) and cloud nodes (CNs) are considered as virtual entities, hosted by the physical devices of the edge tier and cloud tier, respectively. The ENs and CNs' deployment on their respective physical hosts is transparent to the CoS virtualization model, and should be handled by typical cloud and edge computing virtualization models, whose properties we do not discuss in detail here. We follow the principle of overlay virtualization [60], respective to building VNs in an overlay layer, over either PSANs or readily available ENs and CNs (see Figure 3). By being distributed, lacking central control and allowing resource sharing, overlays are ideal candidates for CoS virtualization [60]. In this overlay, application entry points (AEPs) receive applications arriving at the CoS system, which will be served by VNs. A CoS virtualization model, such as Olympus, defines how to provide applications with the resources/services offered by PSANs, ENs and CNs, through VNs.

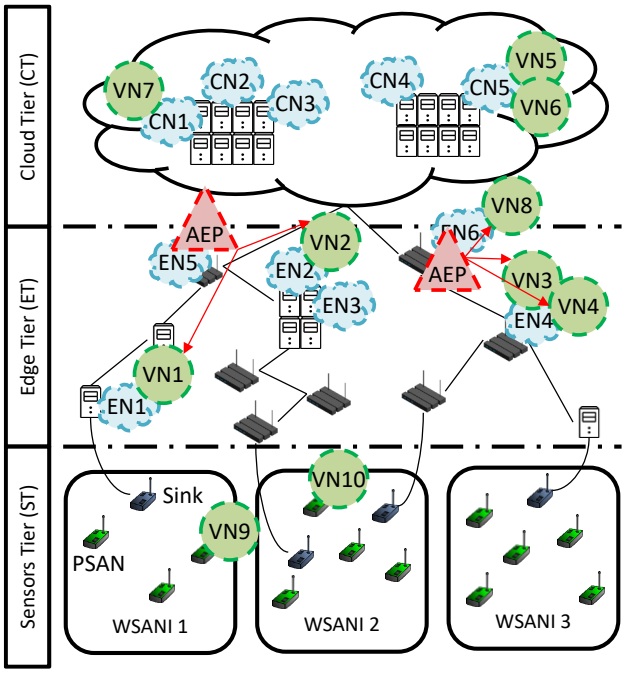

Figure 3. The CoS infrastructure served to applications through an overlay network

In this Chapter, we describe the main aspects of Olympus, to be used as part of an integrated solution for resource allocation in CoS. In Section 3.1, we present an overview of the Olympus virtualization model. More details on Olympus are found in [40]. In Section 3.2, we propose the application model considered in Olympus. Finally, in Section 3.3, we describe a detailed and concrete architecture to put into practice the concepts behind Olympus.

3.1 Overview of Olympus

Olympus is a decentralized virtualization model for CoS. It seeks to make the best use of the cloud and the physical WSAN environments by finding a balance between two possible approaches for executing services: centrally, inside the cloud, and locally, within the physical sensors. Olympus leverages the use of information fusion to ensure that the system will provide data at a given abstraction level more adequate to user applications. Olympus is a decentralized virtualization model since the physical nodes can locally perform the necessary procedures for creating and running the virtual sensor. Therefore, in Olympus, application decision processes are performed partly within physical sensors and partly within the cloud. Olympus abstracts the physical world, by abstracting issues regarding the spatial/geographical distribution of each sensor at the physical layer. Over the physical layer, there is the information fusion layer, based on the information fusion levels according to the classifications of the data-feature-decision (DFD) information fusion model [9].

In Olympus, we make use of the Data-Feature-Decision (DFD) model that provides a classification for information fusion techniques according to the abstraction of the input and output data. In Data In-Data Out (DAI-DAO), information fusion deals with measurement level data as input and output. In Data In-Feature Out (DAI-FEO), information fusion uses data at the measurement level as input to extract attributes or characteristics that describe more summarized information for a given monitored area. In the Feature In-Feature Out (FEI-FEO) category, information fusion works on a set of features to improve or refine an existing characteristic or attribute, or to extract new ones. In the Feature In-Decision Out (FEI-DEO) category, information fusion uses a number of features extracted for generating a symbolic representation (or a decision). In the Decision In-Decision Out (DEI-DEO) category, decisions can be merged to obtain new decisions. Finally, in the Data In-Decision Out (DAI-DEO), either a decision is made directly over raw data, as an atomic procedure, or the information fusion

process under this category can be broken into several atomic parts pertaining to other categories.

In Olympus, an application is considered as a set of services that must be performed to accomplish the application goals. Each application has a finite lifespan and it is interested in a particular geographical area. An application defines a set of QoS requirements, described in terms of maximum end-to-end delay, maximum percentage of packet loss, and energy consumption. Moreover, applications require a set of services provided by the physical WSAN nodes that are described in terms of the following provided services: (i) Data collection, (ii) Processing, (iii) Decision, (iv) Routing, and (v) Actuation. Such capabilities must be published within the cloud in a central repository through a publish/subscribe mechanism. However, physical sensors connected to the CoS must have the minimal capability of locally (in its physical location) providing continuous raw data at a periodic rate. This premise is less restrictive than the works proposing centralized CoS infrastructures support, in which the physical nodes must provide such raw data directly to the sink node.

For connecting applications to physical WSAN nodes, one or more virtual WSAN must be created. A virtual WSAN is composed by providing logical connectivity among the physical nodes. Such physical nodes are grouped into different virtual WSANs based on the phenomenon being monitored or the service being provided. A virtual WSAN node in Olympus is an abstraction of a set of physical nodes, from which the virtual node obtains data. Such a virtual node is considered a computational entity capable of performing a set of information fusion techniques at a given level of DFD model, as shown in Figure 4. Virtual nodes may also form Logical Neighborhoods. In contrast with physical neighborhoods, usually defined in terms of radio ranges, the nodes included in a logical neighborhood are specified by the application based on specific requirements.

In Olympus there is a computational entity, which we term the Virtualization Manager. This entity runs within the cloud and has several responsibilities regarding the model execution management. Our model is said to be partly decentralized because the services allocated by this entity will run within the physical nodes.

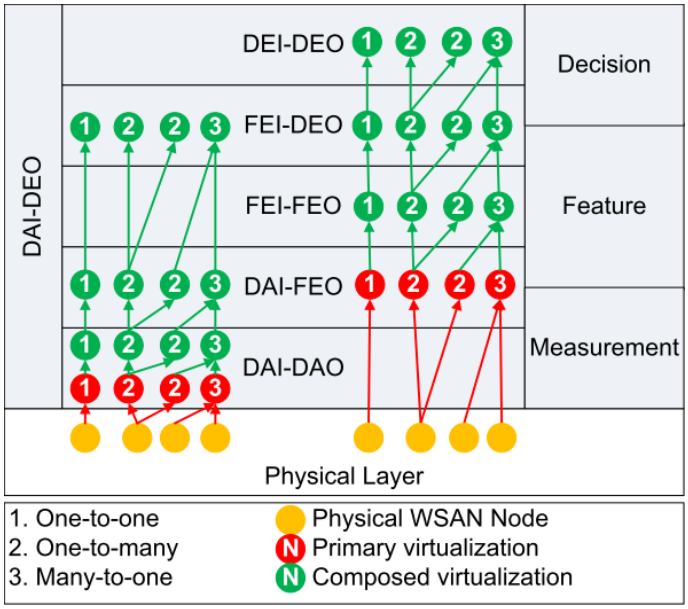

Figure 4. Linking VNs and data abstraction levels of information fusion in Olympus

Each VN in Olympus represents the implementation of an information fusion procedure, which can be reused by several applications. In Olympus, the VNs depend not only on the information fusion level of the input/output data, but also on the source of such data within the physical network. Olympus supports traditional (i) one-to-one, (ii) one-to-many, and (iii) many-to-one virtualization schemes. Moreover, it supports primary virtualization (a virtualization performed when a first VN is instantiated directly from a PSAN), as well as composed virtualizations (in which the virtual sensors are created from another VN).

Olympus operation model comprises both sequential phases: (i) VN creation (instantiation) and (ii) VN operation. In the first phase, Olympus considers a publish/subscribe mechanism to choose the proper PSANs that meet the requirements of the applications requesting the creation of the VN. The software instance of the VN is responsible to read the application requirements (through subscriptions) and the PSAN capabilities published in cloud, for deciding if a new instance of a VN should be created, or an existing instance should be reused. The PSANs, on receiving any request (sent by the Virtualization Manager) to start the instantiation of a VN, are elected as leaders. It is the responsibility of such leader nodes (elected by the Virtualization Manager) to search for and establish routes for communicating with other physical sensors (possibly in other separate physical WSAN) required by a given virtual sensor to be created. Leader nodes will be the reference nodes for the software instance of the VN to communicate with when retrieving data or performing procedures for

VN creation and operation management. Leader nodes are the ones that perform the information fusion techniques at the highest level required by the instantiated VN. Therefore, VN creation is performed partly within the cloud and partly within the PSANs.

When the second phase starts, all PSANs are ready for performing any application request allocated to the respective VNs. In this phase, the software instance of the VN allocates the execution of the application requests within the PSANs (service allocation). During operation, if an undesired state is detected, the software instance of the VN will issue warnings to the application users through push communication. However, every physical sensor must be ready to perform pull communication during the virtual sensor operation management. This is because the application users may want to stay up to date with the current execution status of the allocated services. Olympus supports both time-based (pull model) and event-based (push model) applications simultaneously. If any PSAN pertains to a separate WSANI, then the routing among the PSANs must go through the sink nodes of both WSANIs to enable passage through the cloud. One first approach to resolve this issue in Olympus is to treat the issue as a routing problem. The second possibility is equivalent to the approach of a centralized CoS virtualization model. That is to say, that the PSANs send data to the cloud through the sink nodes, and the VN creation takes place only within the cloud. Finally, the VN operation is more prone to run within PSANs than VN instantiation. VN may even operate without any communication with the cloud. However, there are still procedures of VN operation that may be performed partly within the cloud, such as the case of many-to-one virtualization comprising nodes physically separated in different WSANs. In face of the presented discussions, Olympus is considered a hybrid (partly decentralized) CoS virtualization model.

3.2 Application Model

In this Section, we describe in more details applications and requests, mentioned in Section 1.2. According to Equation (5), we describe each request in terms of the list of predecessor requests (PE), type of request (TY) and lists of non-negotiable and negotiable requirements (NNRR and NRR). Per Equation (6), users describe each application in terms of the time (within the timespan τ) at which an application arrives in its AEP (TI), the value of priority for the application (PI) and the list of requests of the application (LRQ).

$$RQ = \langle PE, TY, NNRR, NRR \rangle \quad (9)$$

$$APP = \langle TI, PI, LRQ \rangle \quad (10)$$

$$(TI \in \mathbb{R}^{+} \mid TI \leq \tau)\ and\ (PI \in \mathbb{R})$$

We model precedence relationships among requests of the same application in terms of the data dependence among requests. Therefore, users must define the list of predecessors (PE) for a request. This list stores the identifiers of other requests from the same application that must run to completion before starting the current request. Based on such information, we model the requests of an application as vertices connected by directed edges without loops, thus forming a directed acyclic graph (DAG) [51]. Figure 5 shows an example of an application with eight requests modeled as a DAG.

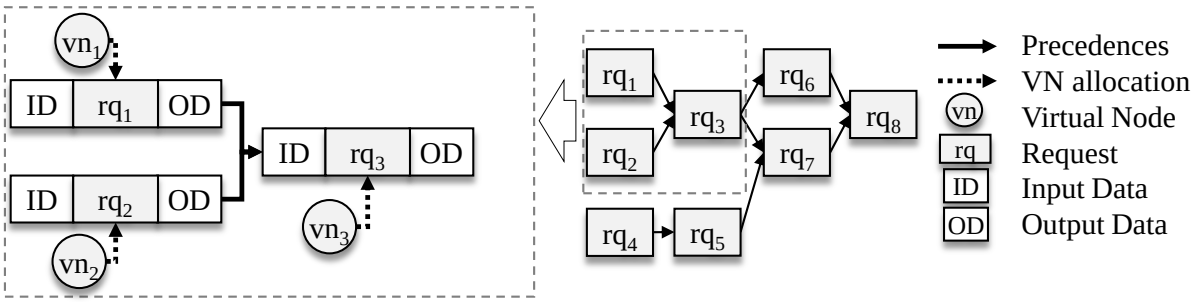

Figure 5. Application modeled as a DAG of requests

In Olympus, we consider both event and time-based applications [12][25]. A time-based application defines the exact moment in time for demanding data from the VN. Thus, the pull model attends time-based applications. In the pull model, a VN coordinates the operation of its underlying infrastructure, to respond to the application, at the time defined by the application. Thus, for each request, we also define TY = {periodic, aperiodic, root, tree} as the type of request. In case of a time-based application, can be periodic or aperiodic, and, in case of an event based application, can be root or tree [12][25]. Aperiodic requests are performed when arriving at the VN and are performed only once. Periodic requests are maintained by a limited number of periods (defined by users) in the VN, since they will be reactivated from time to time. An aperiodic request cannot succeed a periodic request. All the successors of a periodic request are also periodic. Both root and tree requests will wait standby in the VN for a given time t, chosen by the user. Root requests are in the base of the DAG of requests of the event-based application, and are performed always when the VN reports the event expected (through the data provisioning service) within the time t. The Tree

requests also standby in their VNs for the same time t, but Tree requests are the ones from event-based applications that have predecessor requests. They do not wait for events for being performed, just for the output data of predecessor request.

Finally, each request includes lists of non-negotiable and negotiable requirements that must be fulfilled by an allocated VN. Thus, NNRR and NRR are lists that describe such requirements. In case of NRR, the list includes the respective fulfillment level of the requirement. This level is expressed through a percentage, respective to how much the provided data must meet the requirement expectation. In this work, the data freshness is within the list of NRR, and the data type is within the list of NNRR.

Considering that the CoS operates for a given timespan $\tau>0$, an application must arrive at a given AEP at a time TI within this timespan. Upon arrival, an application may be fully performed within the timespan τ. Finally, some applications can be more priority than others can. Therefore, applications have a priority value PI, so that the higher the value of PI is, the application is more priority than others (thus its requests are more priority to be served by the VNs than requests from other applications).

As mentioned in Section 2.7, although task scheduling is one of the challenges raised in this thesis proposing a WSAN task scheduling and execution algorithm for performing the engagement of the underlying CoS infrastructure is not in the scope of our current work. Several algorithms, such as [23][24][54], can be used as solutions to this problem. In this work, we consider that the SAM has a fixed delay for returning its data, which we call the data update time (DUpTm).

3.3 The CoS system architecture: applying the concepts behind Olympus

In this Section, we present the architecture (Figure 6) proposed to put into practice the concepts behind Olympus [40]. Our architecture comprises four subsystems: the CoS Manager Subsystem (CMS), the Application Manager Subsystem (AMS), the Virtualization Subsystem (VS) and the Virtual Node Subsystem (VNS). We consider that CoS virtualization is held within the cloud and edge tiers, thus the VS components are distributed among such tiers. The VS supports the instantiation and execution of multiple VNs, each one running a VNS.

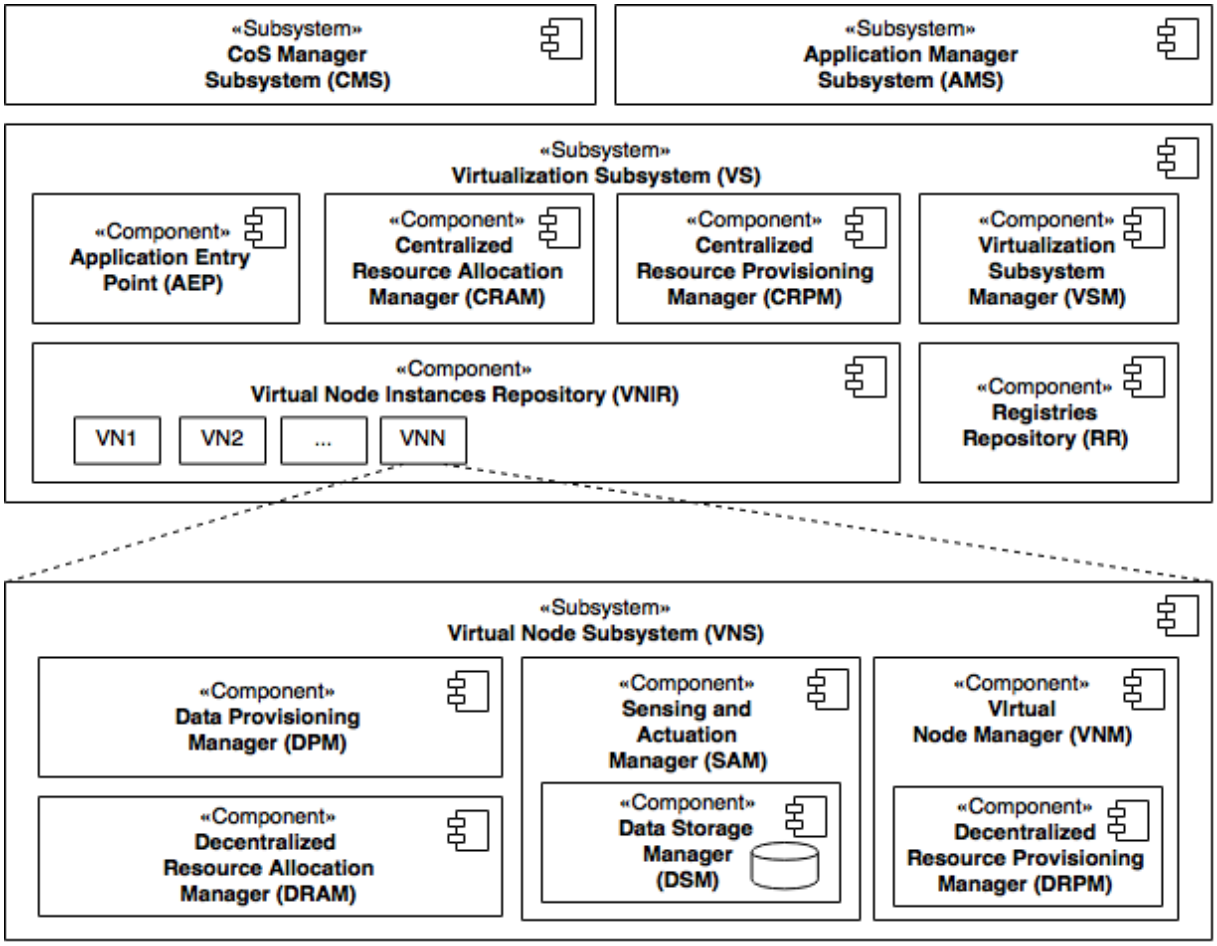

Figure 6. The proposed CoS system architecture

The CMS provides an Application Programming Interface (API) that allows InPs to manage the CoS system. This API allows describing and storing the data types (along with their dependencies and information fusion techniques) handled by the CoS. InPs must consider a desired level of target users' expertise in an application domain, when defining a data type. Finally, the API allows provisioning VN instances proactively (as well as deleting and reconfiguring VNs), according to InPs' criteria.

The AMS has the main functionality of supporting users in the development of applications. It provides an API that allows users to express their requests for the data types existing in the CoS, stored in the CMS. Since the specification of this subsystem is extensive and there are several approaches for developing sensing-based applications, it is out of the scope of our work. Therefore, we suggest methods existing in literature for implementing its main functionality. For discovering and selecting the available data types in the CoS, we suggest using typical service discovery methods [15]. To express and manage application workflows, we suggest using domain specific languages (DSL) or semantic queries [101]. Users

can submit their developed workflows through this API to be performed by the CoS, which will return a final data.

In Section 3.3.1 we describe the VS. In Section 3.3.2, we describe the VNS. In Section 3.3.3, we discuss the key design choices for the CoS system implementation. In Section 3.3.4, we describe the operational sequence of the CoS system.

3.3.1 Virtualization Subsystem

Considering the VS, it comprises six components to support the management of VNs: Virtualization Subsystem Manager (VSM), VN Instances Repository (VNIR), Registries Repository (RR), Centralized Resource Allocation Manager (CRAM), Centralized Resource Provisioning Manager (CRPM) and Application Entry Point (AEP). The VSM is the main component of the VS architecture, and its goal is to coordinate the actions of the other VS' components. When the VS boots, the VSM starts these components. It is a distributed component and runs on both ENs and CNs.

During CoS operation, applications arrive in the CoS through AEPs. An application may arrive, for instance, either through the internet or from a user device (such as a PC or smartphone) connected directly to an EN. Either the CNs or ENs are possible candidates to hold AEPs. The AEP has the goal of handling the arrivals of applications in the CoS, immediately transmitting them to the CRAM. The CRAM and CRPM perform the centralized parts of, respectively, the chosen resource allocation and resource provisioning algorithms. The CRAM is responsible for assigning VNs to application requests. The CRPM is responsible for handling the instantiation and management of VNs.

The VS also comprises the RR, for storing the registries and locations of PSANs and VNs existing in the CoS. The RR can be implemented either in a centralized manner in a CN, or distributed among several CNs and/or ENs. In the latter case, the RR of each EN comprises registries of VNs and PSANs that are within the scope of the local region of the respective EN. Such approach thus, takes advantage of the location-awareness capability of the edge tier, and it is more energy efficient. In the former case, the CN comprises a broader neighborhood area, possibly comprising registries of all existing VNs and PSANs. Regardless, upon receiving a registry from either a VN or a PSAN, the RR decides about disseminating this registry among other nodes implementing the RR. Moreover, the VNIR stores various VN instances locally. Thus, several VNs run within each VNIR.

3.3.2 Virtual Node Subsystem

The VNS comprises six components: the VN Manager (VNM), the Decentralized Resource Provisioning Manager (DRPM), the Decentralized Resource Allocation Manager (DRAM), the Data Provisioning Manager (DPM), the Data Storage Manager (DSM) and the Sensing and Actuation Manager (SAM). The VNM is the first VNS' component to boot, and coordinates the execution of all the other components of a VN. Moreover, it handles the data communication among VNs, forming a virtual network [60], so that VNs can exchange their data among themselves to accomplish the application workflows.

The DRAM and DRPM perform the decentralized parts of, respectively, the resource allocation and resource provisioning algorithms. Moreover, the DRPM boots along with the VNM, and performs the initialization of the VN, storing its configurations. It is also responsible to publish, proactively, and keep updated VN's information in the local RR.

The DPM provides data in response to application requests, also providing values to the negotiable and non-negotiable application requirements, in order to fulfill them. In addition, the DPM provides information regarding the description of its data provisioning service. In a practical situation, we assume the need of using a standard for describing such information, independently on the details of underlying and heterogeneous devices, protocols and formats. Since this standardization is not in the scope of our work, we suggest well-established solutions for it such as the Sensor Modeling Language (SensorML) [19].

The DPM implements the data provisioning service, and can perform a resource allocation decision, accessing either the DSM or SAM for service completion. The DSM manages the local memory of the VN. It provides functions for accessing this local memory and inserting/retrieving historical data from it. The SAM implements the functions of the VN program for performing data updates, and store the recently updated data directly into the DSM. In the next Section we discuss some key design choices to be made, in order to concretize the CoS as a real software system.

3.3.3 Key design choices for the CoS system implementation

Two decisions are fundamental to define the CoS system's operational sequence, implementation and consequently energy consumption. The first decision concerns choosing the degree of decentralization of the resource allocation technique used by the CoS system [97]. Centralized techniques are usually more accurate in decision-making (achieving optimal

solutions) than decentralized techniques. However, they tend to impose a greater communication/energy overhead on the CoS system, since the central decision-making node must obtain information from all the subordinated VNs. As an attempt to balance this trade-off as best as possible, ensuring a greener CoS system design, we suggest using a hybrid algorithm.

Hybrid algorithms comprise centralized and decentralized decision phases. By leveraging the edge tier, it becomes possible to perform the centralized decision phase within an EN, for a localized vicinity of VNs under its scope. Meanwhile, the decentralized decision phase is performed within VNs. Moreover, to further reduce the computation time (and energy) used to make decisions in both decentralized and centralized decision phases, we suggest using heuristic approaches. They are known for being able to achieve near-optimal solutions in reduced computation time for NP-complete problems.

The second decision regards choosing the tier to host and run VNs, and for such we chose the ET. ENs have greater computation and communication capabilities than PSANs, to better manage VNs. Moreover, ENs are in a privileged position, in relation to the CNs, for linking PSANs from different WSANIs under the same VN. Finally, hosting VNs in the same tier of the decision-making ENs shortens the distances between such nodes, saving energy for the CoS by reducing the communication overhead.

Therefore, only ENs will host the VS and the VNS. The CMS and AMS are able to run fully in the CT. They may take advantage of centralized data storages in the cloud to perform their functions. Finally, an application may arrive at an AEP either through the Internet or from a user device (such as a PC or smartphone) connected directly to an EN. In next Section, we describe the operation sequence respective to our design choices.

3.3.4 Operational Sequence of the CoS System

After the development of a hypothetical application workflow, users submit it to the CoS. The AMS will communicate with the AEPs which are close to VNs (or PSANs) that provide the data types required by the application. We assume the adoption of an algorithm, whose specification is out of the scope of this work, to select the best AEP, according to criteria chosen by InPs. The specification of this algorithm, which can be [103] for instance, is out of the scope of this work. Then, the workflow arrives in the selected AEP, which redirects the application to the CRAM at the same EN.

The CRAM makes decisions periodically over a buffered set of applications. When the current buffering period expires, the centralized phase of the resource allocation technique runs. A new VN may need to be instantiated, to serve one of the application requests. In this case, the CRAM calls the CRPM, within the same EN, to perform the centralized part of resource provisioning. The CRPM calls the VNIR (either local or at another EN), to instantiate the VN. The VNIR starts the VNM of the respective VN, and the DRPM also starts its execution in this moment. Then, the VNIR returns the control to the CRPM, which calls the DRPM. The DRPM may perform a decentralized part of resource provisioning, and then start the other VN's components. Finally, the DRPM registers the VN instance, calling the RR locally. Then, it returns the control to the CRAM.

When every request has a selected VN, the CRAM calls the DRAM of each selected VN. If no VN is available to meet a request (and no new VNs can be created), the respective request is refused and the information regarding this refusal is returned to the original AEP. Upon arriving at the DRAM of the respective selected VN, the request is queued, waiting to be served serially by the DPM. The request waits for the arrival of any input data (through the VNM), to free its precedence restrictions. When the request is removed from queue, the DPM is called.

According to the decisions made by the CRAM/DRAM (forwarded along with the request), the DPM may perform a data update or return historical data. In the first case, it accesses the SAM, which will perform the data update and store the fresh data into the DSM. In the second case, the DPM will access the DSM directly, to retrieve the most recent data. Then, the request is met by the DPM and the control flow is returned to the DRAM. If the request has a successor request in its application workflow, its data is transmitted to the VN that was allocated to the successor request. The DRAM passes the control to the VNM, which forwards the message to the VNM of the next VN, who calls the respective DRAM. In other words, this transmission occurs between the DRAM of both VNs. Otherwise, this is the last request of the application workflow, so its data is transmitted back to the original AEP. In this case, the DRAM of the VN that performed the last request of the workflow transmits a message to the CRAM of the VS that hosts the AEP, then the CRAM forwards the message to the AEP. Finally, the data is delivered to the application user through the AEP.

3.4 Conclusion

In this chapter, we described our proposal of an information fusion and decentralized CoS virtualization model, which we term Olympus. Thus proposal contributes by achieving the first goal of this thesis, related to the research question 2.

As mentioned in Section 2.7, Olympus addresses the research gap regading the lack of a decentralized full device virtualization model, with devices playing an active role as resource providers and considering the edge tier in its architecture. In addition, Olympus supports effective, fast and lightweight algorithms to perform resource allocation and task scheduling. Finally, Olympus presents an application model which is able to model priorities and precedencies among requests and tasks, and the sharing of requests and tasks in common among multiple applications and VN programs. Olympus also supports both time-based (pull) and event-based (push) applications simultaneously.

In Chapter 4, we investigate research question 3, and present our specific algorithm to perform resource allocation in CoS, called Zeus.

4 Resource allocation in the Cloud of Sensors

In this Chapter, we discuss our approach for resource allocation in CoS. As in other works in literature [73], the problem of resource allocation in CoS falls within the typical class of mixed integer non-linear programming problems (MINPP) [17], [66]. In this chapter we describe the formulation of our MINPP for resource allocation in CoS. Section 4.1 introduces all the sets and decision variables used in the MINPP formulation. Section 4.2 presents the objective function and constraints of our MINPP.

To solve our MINPP in order to seek the optimal solution, there are a number of methods in the literature, such as the linear programming techniques and their variants [67]. However, our formulated MINPP is an NP-complete problem [46]. The proof of NP-completeness of our MINPP is presented in Section 4.3. To solve, in polynomial time, practical instances of our MINPP with arbitrary sizes [78], we propose a hybrid and heuristic based algorithm, called Zeus. Therefore, Section 4.4 presents Zeus, our proposed algorithm to solve the MINPP for resource allocation in CoS.

4.1 Parameters and variables design

Among the MINPP input parameters, we define the sets in Table 4, and the remaining parameters in Table 5. In addition, we define the sets $SPSAN$, comprising the PSANs, and $SECN$, comprising the hosts of VNs (ENs and CNs).

TABLE 4. SUMMARY OF SETS

Definition	Description	Subscript Indexes
$SPSAN = \{k \mid k \in \mathbb{N}_0, k < K\}$	Set of K PSANs	k
$SECN = \{h \mid h \in \mathbb{N}_0, h < H\}$	Set of H hosts (EN or CN) of VNs	h
$SVN = \{i \mid i \in \mathbb{N}_0, i < I\}$	Set of I VNs	i, c
$SRQ = \{j \mid j \in \mathbb{N}_0, j < J\}$	Set of J requests	j, a, b
$SPRE = \{(a,b) \mid a,b \in SRQ, a \neq b\}$	Set of precedencies (a precedes b)	(a,b)
$SP = \{p \mid p \in \mathbb{N}_0, p < P\}$ $SP^1 = \{p \mid p \in \mathbb{N}, p < P\}$	Set of all P periods, and set of periods starting from p=1	p, q

Moreover, we model a resource provisioning function per Equation (11), implemented by a proactive resource provisioning process, as mentioned in Section 2.2. This resource provisioning function maps the sets SPSAN and SECN to the set SVN, which comprises the VNs defined in Section 2.2. This function also outputs the matrixes of VN x PSAN mapping (VPM_{ik}) and VN x host mapping ($VECM_{ih}$). The $VECM_{ih}$ matrix stores the usage levels (0-100%) of

each host by each VN. The VPM_{ik} matrix stores the usage levels (0-100%) of each PSAN by each VN. Such matrixes are also input parameters to our MINPP.

$$f_{phy \rightarrow vir}(SPSAN, SECN) = SVN, VPM_{ik}, VECM_{ih} \quad (11)$$

In addition, we represent the applications in terms of a digraph $G = (SRQ, SPRE)$ where $(a, b) \in SPRE$ represents that request a must be met before request b. We assume the digraph G only contains immediate precedence relationships, that is, if $SRQ = \{1,2,3\}$ and $\{(1,2),(2,3)\} \subset SPRE$, then $(1,3) \notin SPRE$.

TABLE 5. SUMMARY OF REMAINING PARAMETERS

Definition	Description	Definition	Description
BW_{ic}	Bandwidth between VNs i and c	$WEVRA_i$	limit of energy reserved for the window in VN i
TLo_i	Transmission Load of VN i	WEP_k	limit of energy reserved for the window in PSAN k
PLo_i	Processing Load of VN i	$WEVDU_i$	limit of energy reserved for the window in VN i
$LDUPW_i$	Last data update time of VN i achieved during previous decision window	$Vsup_k$	Sensing supply voltage of PSAN k
$DUpTm_i$	Data update time of VN i	$Isens_k$	total current required for sensing activity of PSAN k
PPw_i	Processing power of VN i	$\xi sens_k$	time duration for sensing component to collect data of PSAN k
PPw_h	Processing power of the host h (EN or CN)	$Eelec_k$	radio energy dissipation of PSAN k
DFT_j	Data Freshness Threshold of request j	εamp_k	amplifier energy dissipation of PSAN k
$VECM_{ih}$	VN i x host h mapping	d_{ik}	Distance between PSAN k and VN i
$Pmax_h$	Maximum power dissipated by host h	$\left[C \times V_{dd}^2 + f\right]_k$	Processing constant of PSAN k
b_{ik}	Amount of bytes requested by VN i to PSAN k	n_{ik}	Number of actuations of PSAN k to VN i
$ETMDW$	End Time of the Decision Window	$Eact_k$	Energy spent for each actuation by VN k

In our MINPP formulation, we consider a time-based decision window, defined by a fixed value of duration ($ETMDW$, in Table 5). The resource allocation decision, achieved by solving the MINPP, is valid for the duration of this decision window. Such a decision must be reviewed whenever any of the sets defined in Table 4 changes (for instance, due to application arrivals), or when the execution of the current resource allocation decision is completed (when the decision window ends). We consider partitioning the decision window into P periods, so that it is possible to allocate VNs to requests within these periods. We refer to such periods through the set SP. Each VN considers the same amount P of periods, and every period

p has a start time, an end time and a duration. However, the times and duration vary among VNs. For instance, a given period p could have different start times, end times and durations in VN 1 and VN 2.

TABLE 6. DECISION VARIABLES

Definition	Description
$x_{ij}^{p} \in \{0,1\}$ $\forall i,j,p \in SVN, SRQ, SP$	Whether VN I is allocated to request j in period p
$y_{i}^{p} \in \{0,1\}$ $\forall i,p \in SVN, SP$	Whether VN i updates its data in period p
$STM_{i}^{p} \in \mathbb{R}$ $\forall i,p \in SVN, SP$	Start time of period p in VN i

Moreover, our MINPP employs three decision variables shown in Table 6: x_{ij}^{p}, y_{i}^{p} and STM_{i}^{p}. The resource allocation decision comprises feasible values of such variables. We define x_{ij}^{p} as binary values to indicate whether to allocate VN i to request j in period p. A request is always fully processed during a single period. When assigning $x_{ij}^{p} = 1$ at any given period p, for a given VN i and request j, the values of x_{ij}^{p} for every period p'>p must also be equal to one, for the same values of i and j. Thus, whenever an allocation occurs, it cannot be undone or re-done during the remainder of the decision window. In addition, the binary variables y_{i}^{p} indicate whether VN i performs a data update in period p. Finally, the real-valued decision variable STM_{i}^{p} indicates the start time of each period p in a VN i.

4.2 MINPP Formulation for resource allocation in CoS

In this Section, we describe the objective function of our problem, defined per Equation (12). We also describe the constraints of our problem, defined per Equations (22) to (34).

$$\max \sum_{j \in SRQ} \sum_{i \in SVN} \sum_{p \in \mathrm{SP}^1} xa_{ij}^{p} * u_{ij}^{p} \tag{12}$$

Where:

$$xa_{ij}^{p} = x_{ij}^{p} - x_{ij}^{p-1} \qquad \begin{matrix} \forall i \in SVN \\ \forall j \in SRQ \\ \forall p \in \mathrm{SP}^1 \end{matrix} \tag{13}$$

$$ETM_{i}^{p} = TTM_{i}^{p} + PTM_{i}^{p} + UTM_{i}^{p} + STM_{i}^{p} \qquad \begin{matrix} \forall i \in SVN \\ \forall p \in SP \end{matrix} \tag{14}$$

$$UTM_{i}^{p} = DUpTm_{i} \times y_{i}^{p} \qquad \begin{matrix} \forall i \in SVN \\ \forall p \in SP \end{matrix} \tag{15}$$

$$PPw_{i} = \sum_{h \in SECN} VECM_{ih} \times PPw_{h} \qquad \forall i \in SVN \tag{16}$$

$$PTM_i^p = \begin{cases} \frac{PLo_i}{PPw_i} \sum_{j \in SRQ} xa_{ij}^p & p > 0 \\ 0 & p = 0 \end{cases} \quad \begin{matrix} \forall i \in SVN \\ \forall p \in SP \end{matrix} \quad (17)$$

$$TTM_i^p = \sum_{\substack{(a,b) \in \\ SPRE}} \sum_{\substack{c \in SVN \\ c \neq i}} \sum_{\substack{q=1 \\ q \neq p}}^{P} xa_{ia}^p \times \frac{TLo_i}{BW_{ic}} \times xa_{cb}^q \quad \begin{matrix} \forall i \in SVN \\ \forall p \in SP \end{matrix} \quad (18)$$

$$LDU_i^p = max\left(\left\{y_i^q \times STM_i^q + UTM_i^q + LDUPW_i \times (y_i^q - 1) | q \in [0..p]\right\}\right) \quad \begin{matrix} \forall i \in SVN \\ \forall p \in \mathrm{SP}^1 \end{matrix} \quad (19)$$

$$DF_i^p = STM_i^p + UTM_i^p - LDU_i^p \quad \begin{matrix} \forall i \in SVN \\ \forall p \in \mathrm{SP}^1 \end{matrix} \quad (20)$$

$$u_{ij}^p = \begin{cases} 0 & (DF_i^p > DFT_j) \\ 1 - \frac{DF_i^p}{DFT_j} & (DFT_j \geq DF_i^p \geq 0) \end{cases} \quad \begin{matrix} \forall i \in SVN \\ \forall j \in SRQ \\ \forall p \in \mathrm{SP}^1 \end{matrix} \quad (21)$$

Subject to:

$$xa_{ij}^p \geq 0 \quad \begin{matrix} \forall i \in SVN \\ \forall j \in SRQ \\ \forall p \in \mathrm{SP}^1 \end{matrix} \quad (22)$$

$$x_{ij}^0 = 0 \quad \begin{matrix} \forall i \in SVN \\ \forall j \in SRQ \end{matrix} \quad (23)$$

$$y_i^0 = 0 \quad \forall i \in SVN \quad (24)$$

$$STM_i^0 = 0 \quad \forall i \in SVN \quad (25)$$

$$\sum_{i \in SVN} \sum_{p \in \mathrm{SP}^1} xa_{ij}^p \leq 1 \quad \forall j \in SRQ \quad (26)$$

$$\sum_{p \in \mathrm{SP}^1} xa_{ij}^p \leq FRR_{ij} \quad \begin{matrix} \forall i \in SVN \\ \forall j \in SRQ \end{matrix} \quad (27)$$

$$\sum_{i \in SVN} \sum_{p \in \mathrm{SP}^1} \left(x_{ia}^p - x_{ib}^p\right) \geq 1 \quad \begin{matrix} \forall (a,b) \\ \in SPRE \end{matrix} \quad (28)$$

$$STM_i^p \geq max\left(\left\{xa_{ib}^p \times STM_c^q \times xa_{ca}^q | (a,b) \in SPRE, c \in SVN, q \in SP, q \neq p\right\}\right) \quad \begin{matrix} \forall i \in SVN \\ \forall p \in \mathrm{SP}^1 \end{matrix} \quad (29)$$

$$ETM_i^p \leq ETMDW \quad \begin{matrix} \forall i \in SVN \\ p = P - 1 \end{matrix} \quad (30)$$

$$STM_i^p \geq ETM_i^{p-1} \quad \begin{matrix} \forall i \in SVN \\ \forall p \in \mathrm{SP}^1 \end{matrix} \quad (31)$$

$$\sum_{h \in SECN} ERA_{ih} \leq WEVRA_i \quad \forall i \in SVN \quad (32)$$

$$\sum_{i \in SVN} \left[EDU_{ik} \times VPM_{ik} \times \sum_{p \in SP^1} y_i^p \right] \leq WEP_k \qquad \forall k \in SPSAN \qquad (33)$$

$$\sum_{\substack{k \in \\ SPSAN}} \left[EDU_{ik} \times VPM_{ik} \times \sum_{p \in SP^1} y_i^p \right] \leq WEVDU_i \qquad \forall i \in SVN \qquad (34)$$

To define our objective function, we define xa_{ij}^p as an auxiliary expression per Equation (13). When we multiply u_{ij}^p by the value of xa_{ij}^p the result is the amount of utility obtained by allocating VN i to request j at period p. As expressed by Equation (12), we want to maximize the sum of all utilities obtained by each allocation. Thus, we make the following considerations regarding our way for calculating utility. First, in our formulation, we obtain values of utility for each period, and as a function of time, which affects data freshness directly. Thus, we consider, for each period p of a VN, besides its start time (STM_i^p), its end time (ETM_i^p), defined per Equation (14), and its duration.

We define the time spent for updating data (UTM_i^p) per Equation (15). If a decision is made for updating data of VN i at period p, this time is equal to the $DUpTm_i$. The time spent for processing (PTM_i^p) defined per Equation (17), is the time necessary for a VN i to perform its internal procedures in order to meet a request j. Each allocation of VN i to a request j generates the standard processing load in the VN i (PLo_i). In addition, the VN i has a processing power (PPw_i), which is defined per Equation (16). It is a fraction of the processing power of the host (PPw_h), defined by $VECM_{ih}$. It is important to mention that no processing occurs in period zero, since no allocation is possible in this period. Finally, per Equation (18) we define time spent for transmission (TTM_i^p). When VN i is allocated to request "a", the output data must be transmitted to all VNs c that are allocated to the successors b of request "a". Each of such output data transmissions takes the time equal to the rate between TLo_i and BW_{ic}. Equation (18) calculates, for a VN i, the sum of the times spent for each transmission.

Moreover, we calculate the moment in time of the last data update of a VN i at a given period p (LDU_i^p) per Equation (19). This is the highest time value, among the current period p and previous periods of the current decision window, at which a VN i performed a data update. If no data update decision happens during the current decision window, the moment of the last data update considered will be the one occurred in the previous window ($LDUPW_i$), which

is provided as input parameter for the problem. Then, we formally define data freshness per Equation (20). The data freshness of a given VN i in period p is the difference between two moments in time. The first is the moment during current period p in which a request will be served by VN i, and the second is LDU_i^p. In the best case, the data has just been updated in VN i during current period p, thus both moments are equal and data freshness is the best (zero).

Finally, we formally define utility per Equation (21). We assume that the utility decreases linearly and has the maximum and minimum values of 100% and 0%, respectively. We define that, whenever the data freshness (DF_i^p) of a given VN i, in a given period p, is zero, the utility is maximum (100%), i.e. a data update happened in such a period. We assume that values of DF_i^p that are higher than the data freshness threshold of request j (DFT_j) are useless, thus utility is minimum (0%) in this case.

By the constraint defined per Equation (22), once an allocation of a VN to a request occurs, it cannot be undone or re-done during the remainder of the decision window. Moreover, since our objective function has a recursive term, it is necessary to consider period zero as a dummy period. In period zero, we disallow allocations and data updates by VNs. In addition, the start time of period zero is set to the start time of the current decision window, which is time zero (a relative time). We express such statements by constraints, respectively defined per Equations (23), (24) and (25). Per Equation (26) we ensure that requests are performed only once and by a single VN i during the decision window. Per Equation (27) we ensure the allocation of VNs to requests only when VNs match the defined non-negotiable requirements of the requests. The matrix of non-negotiable requirements (FRR_{ij}) has the value of one set at each position where a VN i meets the non-negotiable requirements of request j, and zero otherwise. Per Equation (28) we ensure that whenever a VN is allocated to a request b in a given period, all the “a” predecessors of b must also have VNs allocated, at previous periods. Finally, the constraint defined per Equation (29) ensures that a request b should never start at a time prior to the start of its predecessor request “a”. By Equation (30) we ensure that the end time of the last period of each VN is smaller than the end time of the decision window ($ETMDW$). By Equation (31) we ensure values of start times so that two consecutive periods will never overlap. In addition, the successor period will never happen at a prior moment in time, in relation to a predecessor period.

Finally, constraints in Equations (32), (33) and (34) regard the energy costs of allocating each VN to requests. Such energy is fully consumed by the ENs/CNs that host each VN. Thus, we need a model for energy consumption at edge/cloud tiers. It is important to mention that our MINPP is agnostic to any specific energy model. In this paper, we adopted the same energy model used in our previous work, which targets the edge and cloud paradigms [12]. We described this energy model in detail in Appendix A. Moreover, we implemented our MINPP for resource allocation in CoS, considering the energy model used in this thesis, in the CPLEX software [100].

4.3 Proof of NP-completeness

In this Section, we present a theorem formulated as the statement: “The MINPP for resource allocation in CoS is NP-complete”. It is important to prove this theorem in order to justify the elaboration of strategies based on heuristics for solving our problem. We prove the NP-completeness of our MINPP by restriction, thus showing that it contains a known NP-complete problem as a special case [53]. We consider the well-known multiple knapsack problem (MKP), which is known to be NP-complete [46], as the reference special case, and describe it as follows.

In combinatorial optimization, the traditional MKP considers a set N of items and a set M of knapsacks. Each item j ∈ N has an associated profit (or utility) pj and weight (or cost) wj. Each knapsack i ∈ M has an associated capacity Wi. The objective of the MKP is to maximize the total profit of allocating items to knapsacks, according to the objective function in Equation (35), while the total weight of the chosen items at each knapsack does not exceed the knapsack capacity Wi, according to Equation (37). The decision variable xij is used to represent the allocation of item j to knapsack i, and is binary, according to constraint in Equation (36).

$$\max \sum_{j \in N} \sum_{i \in M} x_{ij} p_j \tag{35}$$

Subject to:

$$\sum_{i \in M} x_{ij} \leq 1 \qquad \forall j \in \mathrm{N} \tag{36}$$

$$\sum_{j \in N} w_j x_{ij} \leq W_i \qquad \forall i \in \mathrm{M} \tag{37}$$

To prove our theorem, we specify additional restrictions to the MINPP, reducing it to the MKP. Let us consider the particular instance of the MINPP characterized by the following setting:

1) Requests have no precedencies, i.e., every application comprises a single request. In other words, we consider $SPRE = \{\emptyset\}$. As a consequence, $TTM_i^p = 0$ $(\forall i, p \in SVN, SP)$ from Equation (18). In addition, decisions become able to be taken for a decision window comprising a single period p. Then the index p can be dropped from our problem, and our decision variable xa_{ij}^p can be reduced to xa_{ij}, representing allocations of this single period. Constraint from Equation (26) comes to the form of Equation (39), analogous to Equation (36).

2) Decisions of data update follow the same pattern of xa_{ij}, i.e. $y_i = xa_{ij}$. Thus, DF_i is zero for every VN, and utility is always maximum for every request. This allows us to drop index i from utility. Therefore, our objective function from Equation (12) comes to the form of Equation (38), analogous to Equation (35).

3) Besides the aforementioned restrictions, we consider L=1 (so that SECN has only one EN or CN) and that all VNs are allocated to this EN or CN, and that this EN or CN has $P_{il} = 1$. Thus, constraint from Equation (32) comes to the form of Equation (40), analogous to Equation (37).

4) The remainder constraints from our MINP are discarded by considering the aforementioned restrictions, along with the following ones. Every VN attends to the non-negotiable requirements of every request (only one data type exists in CoS), $ETMDW \to \infty$, $DUpTm_i = 0$ $(\forall i \in SVN)$, and $SPSAN = \{\emptyset\}$.

$$\max \sum_{j \in SRQ} \sum_{i \in SVN} xa_{ij} * u_j \tag{38}$$

Subject to:

$$\sum_{i \in SVN} xa_{ij} \leq 1 \qquad \forall j \in SRQ \tag{39}$$

$$\sum_{j \in RQ} \frac{PLo_i}{PPw_i} \times xa_{ij} \leq WEVRA_i \qquad \forall i \in SVN \tag{40}$$

Therefore, the restricted MINPP matches the NP-complete MKP. By restriction, also our MINPP must be NP-complete, having the implications that follow. Traditional centralized linear programing algorithms, such as the simplex algorithm, are able to solve our MINPP optimally, after some linearization transformations [79]. A possible approach is to implement such centralized algorithms in the cloud tier, with a global view of all VNs and requests. However, due to the NP-completeness, we cannot expect to be able to solve optimally, and in polynomial time [78], practical instances of our MINPP with arbitrary sizes. Depending on the size of an instance of our MINPP or the available CPU speed, we will often have to be satisfied with computing approximate solutions through heuristic algorithms.

4.4 Zeus

This Section describes Zeus, our hybrid and heuristic-based algorithm to solve the MINPP for resource allocation in CoS. Zeus can be implemented either in the edge or in the cloud tier. Thus, the scope of the resource allocation can be either more global (when implementad at the cloud) or more localized (implemented at the edge), affecting the size of the MINPP in terms of the amount of entities considered. Depending on the size of an instance of the MINPP and the available CPU speed, the time required to find optimal can be prohibitive, especially for delay-sensitive applications. This justifies the elaboration of Zeus as an heuristic algorithm, to find sub-optimal solutions in polynomial time for our problem.

In Zeus, decisions regarding x_{ij}^{p}, y_{i}^{p} and STM_{i}^{p} are taken separately for each request at a time, and are valid for the moment in which they are taken. Therefore, in Zeus formulation we do not consider the decision window mentioned in Section 4.1. Moreover, the fast execution of Zeus (in polynomial time) allows it to run as an online solution for resource allocation. Therefore, Zeus operation is interlaced with the execution of application requests. Finally, Zeus is a hybrid algorithm, and, thus, comprises two phases. The first phase is centralized and can be implemented in the AEPs. The function denoting this phase is f_{cen}, (Table 7). In this phase, the decision regarding x_{ij}^{p} is taken. The second phase is decentralized and implemented in each VN. The respective function is f_{dec}, shown in Table 8. In this phase, decisions regarding y_{i}^{p} and STM_{i}^{p} are taken. We describe the first phase of Zeus in Section 4.4.1, and the second phase in Section 4.4.2. Finally, Section 4.4.3 provides an assessment of the computational complexity of Zeus.

4.4.1 First phase

The first phase of Zeus, in Table 7, starts by defining the empty unique DAG (UDAG) structure (line 2) and running a timer (line 3), whose time buff_tm is provided as a parameter to f_{cen}. The f_{cen} runs periodically, and each cycle is described by the block between lines 4 and 40. In summary, f_{cen} waits for a time duration (fixed for all cycles) for buffering the arriving applications. Simultaneously, requests in common of buffered applications are merged according to their requirements. From lines 6 to 23, we describe our UDAG formation algorithm, which is inspired by [54]. The main difference is that we consider building a UDAG of requests, which differ from tasks as considered in [54]. Another difference is that we consider negotiable (data freshness) and non-negotiable (data type) requirements in the UDAG formation.

TABLE 7. FIRST PHASE OF ZEUS

```
Input: buff_tm, SVN, rule_fcen, rule_fdec
Output: decision x
 1        def fcen(rule_fcen, buff_tm, SVN): // 1st PHASE
 2          UDAG ← {∅};
 3          buffering_timer.start(buff_tm);
 4          while True:
 5            // Begin UDAG formation
 6            for each app in new_apps_arriving():
 7              for each rq in app.getDAGnodes():
 8                related_urq_found ← False;
 9                for each urq in UDAG:
10                  if cmp_non_neg_rr(rq, urq):
11                    urq.update_more_restr_neg_rr(rq);
12                    urq.add_related_rq(rq, rq.app_id);
13                    rq.set_related_urq(urq);
14                    related_urq_found ← True;
15                if !related_urq_found: UDAG.addurq(rq);
16              for each pr in app.getDAGedges():
17                related_upr_found ← False;
18                for each upr in UDAG:
19                  pre_found ← (pr.DAGPre == upr.pre);
20                  pos_found ← (pr.DAGPos == upr.pos);
21                  if !(pre_found and pos_found):
22                    related_upr_found ← True;
23                if !related_upr_found: UDAG.addupr(pr);
24            // End UDAG formation
25
26            if buffering_timer.expired():
27              // Begin deciding xijp
28              pairs ← {∅};
29              for each urq in UDAG:
30                for each v in SVN:
31                  if eval_non_negotiable_rr(urq,v):
32                    pairs += {(urq,v)};
33              x ← choose_best_pairs(pairs, rule_fcen);
34              // End deciding xijp
35              for each urq in UDAG(): urq.fill_tVN(x);
36              for each urq in UDAG():
37                urq.fill_suc_tVN(UDAG);
38                dispatch(urq);
39              UDAG ← {∅};
40              buffering_timer.reset();
```

In line 6, the AEP accesses all applications arriving on it at the beginning of the current cycle of f_{cen}. The AEP iterates for each request (rq) of each application (line 7), to assess if either it should generate a new unique request (urq) in the UDAG (line 15), or it should merge the rq to an existing urq (lines 10-14). The function cmp_non_neg_rr (line 10) returns true if the non-negotiable requirements of rq and urq are the same. In line 11, the function update_more_restr_neg_rr chooses the values of negotiable requirements that are more restrictive between the urq and the rq being merged. We use the max function, thus the new data freshness of the urq is always the highest value between the rq and urq. In line 12, the

urq stores the identifier and information of the rq that is being merged to it. Moreover, in line 13 the rq keeps track of which urq it was merged to. Next, the AEP iterates for each precedence relationship (edge) pr in the graph of each application (line 16), similarly to the procedure of adding urqs to the UDAG. The AEP iterates through the existing upr in the UDAG (line 18), and if no existing edge is found in the UDAG, the AEP adds a new upr to it (line 23). In our work, edges have no requirements to be merged. Moreover, since the AEP stores, in the rq, the information about its related urqs, the comparisons performed in lines 19, 20, 21 are trivial.

After buffering all arriving applications in the current cycle, the AEP checks if its timer has expired, starting a new cycle otherwise. If the timer has expired, the AEP starts making its decision regarding variable x_{ij}^{p} (between lines 28 and 33). From lines 28 to 32, pairs of rqs and VNs are formed. In line 31, function eval_non_negotiable_rr filters all the pairs of rqs and VNs, to comprise only the pairs which meet the non-negotiable requirements. Line 33 calls the function choose_best_pairs, which considers a given rule (parameter rule_fcen), to decide which pairs of rqs and VNs will be in fact performed. The result of this decision is stored in variable x (analogous to x_{ij}^{p}). For instance, we can consider two rules. The first is the energy balancing rule (EBR), which consists of allocating VNs with the highest remaining energy values to each rq. The second is the queue time reduction rule (QTRR), which consists of allocating VNs with the shortest queues to each rq. Among the several rules that could be formulated for deciding x, we chose EBR and QTRR for the following reasons. First, in the context of WSANs applications, due to the energy constrained nature of the nodes [89][90][91][92], a main concern arises regarding how to extend the WSAN lifetime. We covered this by formulating the EBR rule. Second, in the context of edge applications, which require strict response times [39], the main concern regards reducing the response time of applications. Since this response time is mainly influenced by queues formed in the CoS system, we formulated the QTRR.

Line 35 fills each urq with the information regarding the VN allocated to it. Lines 36 and 37 fill each urq with information regarding the VNs allocated to the successors of the urq. The procedure of line 38 transmits the urq to its VN, where it will be handled by f_{dec}. We assume that all requests have at least one feasible pair of rq and VN. Thus, line 39 erases the

UDAG, and resets the buffering timer in line 40. Section 4.4.2 describes the second phase of Zeus.

4.4.2 Second phase

The second phase of Zeus is shown in Table 8. It starts by obtaining the reference to the data provisioning service of the VN (line 2), where the data provisioning service is started. This service is able to provide data to one request at a time. In line 3, the queue of unique requests of the VN is started, empty. The algorithm of f_{dec} runs periodically, and each cycle is described by the block between lines 4 and 15.

TABLE 8. SECOND PHASE OF ZEUS

INPUT: BUFF_TM, SVN, RULE_FCEN, RULE_FDEC OUTPUT: DECISIONS Y AND STM	
1	**def** fdec(rule_fdec): // 2nd PHASE
2	dp = data_provisioning_service();
3	queue ← {∅};
4	**while** True:
5	**for each** urq **in** new_urqs_arriving():
6	add_to_queue(urq);
7	**if** dp.completed():
8	transmit_data_to_suc_VNs(dp.cmpl_urq);
9	dp.is_idle = True;
10	**if** (dp.is_idle) **and** (queue.len > 0):
11	// Begin deciding yip and STMip
12	pri_urq = queue.get_top();
13	y ← decides_for_dupd(urq, rule_fdec);
14	pri_urq.stm = now();
15	dp.provide_data(urq, y);

The VN iterates for each urq arriving from different AEPs in line 5. Line 6 adds each urq to the VN queue, which is a first in–first out (FIFO) queue of unlimited size in this work. When removing an urq from the queue, the VN considers the precedence restrictions among urqs. Thus, it is avoided that the next urq to be removed from queue with a precedence restriction (not able to be performed) blocks the execution of another urq in the middle of the queue and without precedence restrictions. If the data provisioning service of the VN has finished providing data to an urq (line 7), the VN transmits the output data to all the VNs allocated to the successors of current urq (line 8). Due to such procedure, data incoming from other VNs (input data of an urq in the queue) may also arrive in the current VN. Therefore, the VN fills the respective urq with the incoming data and removes the corresponding precedence restriction. If the urq has no successor, the VN transmits data back to the AEPs from which the applications respective to the urq arrived. In line 9, the data provisioning service is set to idle.

Next, the VN checks if the data provisioning service is idle, and the queue is not empty (line 10). Then, per line 12, the VN obtains the next urq from the queue with no precedence restrictions. Then, the VN decides if it should perform a data update (line 13), considering a given rule (parameter rule_fdec). For instance, we consider two rules. First, the avoid negative utility rule (ANUR) updates data whenever the current data freshness provided by the VN is greater than the data freshness threshold of the urq. Second, the maximum utility rule (MUR) consists of always performing the data update. Among the several rules that could be formulated for deciding y (analogous to y_i^p), we chose ANUR and MUR for the following reason. These two rules represent two extremes of all possible decisions regarding utility. MUR allows obtaining the maximum possible utility for all applications, while ANUR ensures obtaining at least the minimum acceptable utility for all applications.

Next, line 14 sets the start time of the urq to the current time of the VN. In line 15, the function provide_data starts processing the urq, setting the value is_idle of the data provisioning service to False. Finally, it is important to mention that our proposal is based on a set of user-configurable rules (rule_fcen and rule_fdec) for making decisions. Therefore, the user may dynamically change the rules during the CoS operation, disseminating new rules to AEPs and VNs.

4.4.3 Computational complexity of Zeus

Let A be the number of applications provided as input to Zeus. Moreover, let R and P be, respectively, the number of requests and precedencies per application. Similarly, let U and Y be, respectively, the number of unique requests and unique precedencies existing in the UDAG. In the worst case (when the UDAG formation algorithm does not merge any request), the maximum values of U and Y are U=A*R and Y=A*P. Finally, let V be the number of VNs provided as input to Zeus.

Zeus algorithm is solvable in polynomial time if its running time is upper bounded by a polynomial expression such as Equation (43), which is built as a function of the size of the data inputs required by Zeus. In other words, Z(A,R,P,U,Y,V) is built as a function of the number of basic operation required by each of the phases of Zeus. We assume that every sentence of Zeus that does not depend on the size of the input data takes a unit (one) computational step. We assume the worst cases when necessary during the calculation of Zeus computational complexity. For instance, we assume that all if statements always result in true, so that their

nested blocks of instructions are always performed. We assume that U and Y are static and equal to their maximum values during the whole execution of the algorithm, although they start at zero and are incremented at each iteration by the loops during the UDAG formation algorithm in reality. Moreover, we assume that more efficient search and sort algorithms could be used for replacing blocks 8-15 and 17-23, which would be performed in complexity O(n*log(n)), such as the binary search, instead of O(n*n).

We formulate Zf_{cen} as follows. Block 1 (lines 9 to 14) costs U*6. Block 2 (lines 7 to 15) comprises Block 1 and costs R * (1 + 1 + (U * 6) + 2). Block 3 (lines 18 to 22) costs Y * 5. Block 4 (lines 16 to 23), which comprises Block 3, costs P * (1 + Y * 5 + 3). Therefore, for each cycle of the algorithm, the UDAG formation procedure (Block 5, from lines 6 to 23, and comprising Blocks 2 and 4) costs A * [(R * 4 + U * R * 6) + (P * 4 + P * Y * 5)].

Block 6 (lines 29 to 32) costs U * V * 4. Line 33 costs U * V * 3. Line 35 costs U * U * 2. Block 7 (lines 36 to 38) costs U * Y * 3. Therefore, Block 8 (lines 26 to 40), comprising Blocks 6 and 7, and lines 33 and 35, costs 4 + U * V * 7 + U * U * 2 + U * Y * 3. Finally, Zf_{cen}, shown in Equation (41), is the sum of the costs of Blocks 5 and 8, plus the cost of lines 1, 2 and 3, which run once in every case of Zeus execution.

We formulate Zf_{dec} as follows. Lines 43, 44 and 45 cost 3. In the worst case, Zf_{cen} allocated the same VN for all urqs, and a typical sort algorithm (such as insertion sort) is used to set up the queue. Thus, Lines 46 and 47 cost U * U. Block 9 (lines 48 to 56) considers the worst case in which every urq has Y successors, thus line 49 iterates for every precedence. In addition, Block 9 considers that the rule to decide f_{dec} iterates for all U. Therefore, Block 9 costs (6 + Y + U). Finally, Zf_{dec}, shown in Equation (42), is the sum of the costs of Lines 42 to 47 and Block 9.

$$Zf_{cen} = \mathrm{A}[(4R + 6UR) + (4P + 5PY)] + (4 + 7UV + 2UU + 3UY) + 3 \quad (41)$$

$$Zf_{dec} = 9 + UU + Y + U \quad (42)$$

$$Z(A, R, P, U, Y, V) = Zf_{cen} + Zf_{dec} \quad (43)$$

$$Z(n) = 17n^4 + 7n^3 + 10n^2 + 16 \quad (44)$$

If we consider that U=A*R and Y=A*P, and that the amount of applications, VNs, requests per application and precedencies per application are the same i.e. A=V=R=P=n, we simplify Equation (43) to Equation (44). Therefore, we assume, in the worst case, the complexity $O(n^4)$ for Zeus. We also estimated the complexity in the best case (all requests are merged, U=R and Y=P), which is $O(n^2)$. Thus, Zeus is solvable in polynomial time. Polynomial

time algorithms are said to be “fast”, in relation to several other classes of algorithms [78]. Therefore, Zeus is able to run as an online solution which makes use of the edge tier. Such characteristics enable Zeus to support delay-sensitive applications.

4.5 Conclusion

In this Chapter we achieved the second goal of this thesis, related to the research question 3. We proposed a formulation to the problem of resource allocation in CoS, along with a heuristic and hybrid algorithm for solving it, which we named Zeus. We modeled the problem of resource allocation in CoS as a typical mixed integer non-linear programming problem (MINPP).

As mentioned in Section 2.7, our proposal addresses the research gap regading the lack of formulations to the problem of resource allocation as a MINPP, considering the maximization of the freshness of the data provided to applications. Zeus addresses the lack of hybrid and heuristic algorithms, which are fast and lightweight, to find near optimal solutions to the problem of resource allocation. Moreover, Zeus handles precedencies among requests and tasks, and the sharing of requests and tasks in common among multiple applications. Finally, Zeus supports time-based applications and a proactive resource provisioning process.

The next step in our research is to evaluate Zeus, with respect to assessing the contributions of Olympus and Zeus mentioned in Section 1.5. In Chapter 6 we describe the evaluation on Zeus.

5 Evaluation

In this Chapter we describe the experimental evaluation performed to assess our work and show that it properly achieves the intended goals. To help understanding our proposal and supporting the scenario used in our evaluation, we present an illustrative and hypothetical example in the application domain of SHM for smart buildings (Section 5.1). In Section 5.2 we describe the metrics used for assessing the performance of Zeus in our experiments. Four experiments were performed with the goal to ascertain whether the contributions intended with the adoption of Zeus were effectively achieved. The first experiment (described in Section 5.3.1) aims at assessing Zeus scalability, both in terms of the number of VNs and the number of applications executed in the CoS. The second experiment (Section 5.3.2) assesses how the use of the edge tier in Zeus allows supporting delay-sensitive applications, in comparison to an approach using only the cloud tier. Moreover, Zeus identifies tasks that are common for multiple applications, performing them only once and sharing the outcome among these multiple applications. Such a feature contributes for improving the WSANs lifetime, and this is assessed in a third experiment (described in Section 5.3.3). The fourth experiment evaluates the quality of the solutions of our MINPP found by Zeus, and shows how much energy it can save for the WSANs when reusing data among multiple applications (Section 5.3.4). Finally, in Section 5.4 we made some considerations on the implementation of the CoS as a real software system, using the FIWARE platform [104].

5.1 Illustrative example

The goal of this illustrative example, based on [22], is to implement a CoS system for monitoring the structural health of buildings in a smart city. Among the requirements of a smart building, assuring the safety of its residents is a key issue, especially considering seismic events. In cities prone to frequent seismic tremors, managers close buildings for months to undergo a detailed manual inspection, to ensure the safety of people. Such an approach is inefficient, resulting in frequent inoperative periods of the building, and consequent economic losses.

5.1.1 CoS infrastructure deployment

In this hypothetical scenario (Figure 7), let us consider a smart building composed of four floors, covering a rectangular area of 50x50 meters and with three meters of height between the floors. The smart building has six PSANs deployed on each floor. All the PSANs are MICAz motes [80][98], whose radio supports 2.40-2.48 GHz band and 250 kbps data rate. MICAz motes include the main board, with processor, radio, memory and batteries. Per floor, five PSANs are connected to a MTS400 board equipped with accelerometer, barometric pressure, ambient light, relative humidity and temperature sensing units. Finally, the remaining PSAN per floor has no sensing units, and is connected to an actuator, a stoplight-style signal. It must have its values set according to the current structural soundness. The common energy source of MICAz motes (two AA batteries) provides up to 16 kJ of energy in a real situation [62]. All the PSANs deployed in the smart building pertain to the same WSANI. Other relevant parameters regarding the sensors tier are summarized in Table 9. Most values in Table 9 were retrieved from our previous works [12][22] and adapted to the current scenario when necessary.

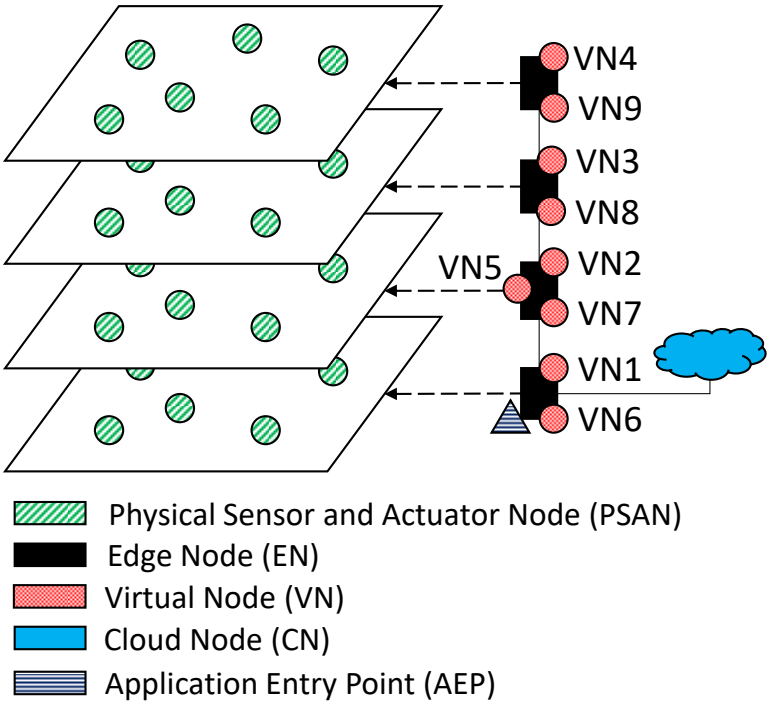

Figure 7. Deployment of the three-tier CoS infrastructure in the smart building

In our hypothetical scenario, the PSANs are programmed in NesC language, under the TinyOS development environment [82], version 2.1. Further information on the configurations

used in the physical and MAC layers can be found in [22]. The InP deploys the PSANs in key locations where the shock response of the structure is the highest. We randomly define such key locations according to the following procedure, also used in [12]. Given the coordinates of the centre of a floor, the "PSAN deployment radius" parameter in Table 9 defines a circle area, centred at the floor centre, within which all the PSANs of the floor are randomly (uniform probability) positioned.

TABLE 9. PARAMETERS OF THE COS INFRASTRUCTURE

Parameter	Value	Source
Sensors Tier		
$Vsup_k$	1.0 V	
$Isens_k$	1 mA	
$\xi sens_k$	0.5 sec	
$Eelec_k$	50.0/(10**9)	
εamp_k	10.0/(10**12)	[12], [22]
$\left[C \times V_{dd}^2 + f\right]_k$	7.310e-10 J/bit	
n_{ik}	1 actuation	
$Eact_k$	0.02 J	
PSAN deployment radius	25m	
Edge Tier		
EN core speed PPw_h	3 x 10^6 cycles/sec	
Number of cores of each EN	4 cores	
Average bandwidth of ENs BWd_{ic}	(10.0**6) bps	[12]
$Pmax_h$	20.0 W	
Transmission delay lambda	0.2 sec	
Cloud Tier		
CN core speed PPw_h	6 x 10^6 cycles/sec	
Number of cores of each CN	128 cores	
Average bandwidth of CNs BWd_{ic}	(10.0**6) bps	[12]
$Pmax_h$	100.0 W	
Transmission delay lambda	0.2 sec	

Besides PSANs, we consider the deployment of gateway nodes of the WSANI in the same corner of each floor, each one attached to a local desktop, composing the ENs (one per floor). Each PSAN is directly connected to the EN of its floor with one-hop distance. The ENs connect all floors through the building Local Area Network (LAN), and to the Internet by using existing broadband internet connections (for instance, ADSL, VDSL, Satellite) in the building, thus being able to reach the CN.

5.1.2 VNs description

In our example, we consider nine data types (numbered as 1 to 9), and nine VNs, one VN to provide (and numbered respectively to) each data type. In the simulations, we choose

the edge tier as the best location in our three-tier CoS architecture to run the VNs, according to the properties of this tier assessed in our previous work [12]. The ENs are in a privileged position, in relation to the CNs, for linking PSANs from different WSANIs under the same VN, and for reducing the latency for time sensitive applications. Relevant parameters of VNs are shown in Table 10.

In line with a proactive resource provisioning process (see Section 2.2), the InP instantiates VN 1, 2, 3 and 4 (Figure 7), one per floor, that provide to applications the cooperative damage coefficients (CDCs) of the respective floor. The CDC [22] is a representation of local decisions made by each PSAN. For each floor the CDC is calculated by the five PSANs with sensing units, in their respective positions, each one performing procedures as described in [22], and coordinated by the VN. Thus, the respective VN stores all the CDCs of a floor. As in [22], we consider a 10-bit CDC for each PSAN. Thus, the data types 1, 2, 3 and 4 are, each, a list of tuples. This list contains one tuple per PSAN with sensing units in the floor, of the format <CDC_value, location_in_floor>.

TABLE 10. PARAMETERS OF COS VIRTUALIZATION OBTAINED FROM [22]

Parameter	Value
$DUpTm_i$ (data type 5)	0.025 sec
$DUpTm_i$ (data types 6-9)	0.125 sec
$DUpTm_i$ (data types 1-4)	0.500 sec
b_{ik} (data type 5)	0 bytes
b_{ik} (data types 6-9)	1 byte
b_{ik} (data types 1-4)	1024 bytes
Output data size (data type 5)	1 byte
Output data size (data types 6-9)	1 byte
Output data size (data types 1-4)	10 bit * 5 PSAN with sensing unit

In addition, the InP implements a VN for providing data type 5, which is the output of the information fusion techinque for deciding about the existence of damage in the building, described in [22]. Data type 5 uses data types 1, 2, 3 and 4 as input. The VN5 has no subordinated PSANs, and is able to perform its function fully within its host EN. Data type 5 has the format of a Boolean variable. Moreover, the InP implements VNs for providing data types 6, 7, 8 and 9, and performing actuation on each floor, depending on data type 5. Such data types consist of actuation feedback (i.e. actuation acknowledgement and logs) retrieved from the physical actuator, in the floor respective to the VN. The InP defines that, when the VN performs a data update, the occurrence of an actuation on the physical environment is implicit. This physical actuation is part of the information fusion technique implemented by

the InP within the VN. Therefore, if several requests access simultaneously this VN, the VN only needs to physically actuate once, and provide the same "actuation feedback" data to all the simultaneous requests.

Finally, we consider that every PSAN is dedicated exclusively to an assigned VN (the VPM_{ik} matrix has only 0 and 1 values, and no sharing of a PSAN by VNs occurs). In addition, whenever a host EN or CN is shared among VNs, the matrix $VECM_{il}$ is filled with equal portions of capacity of the host for each VN sharing it.

5.1.3 Applications description

In our illustrative example, we consider the arrivals of several applications that can be described by the same workflow structure, based on [22]. In Table 11, we show relevant parameters of the applications. Each application has nine requests, each one requesting one of the data types 1-9, and numbered according to the requested data type. The arrival time of applications is randomly and uniformly distributed in between the start time of simulation (simulation time = 0) and the total simulation time (duration). When the application arrives in the CoS, it is immediately assigned to its AEP.

TABLE 11 RELEVANT PARAMETERS OF APPLICATIONS

Parameter	Value	Source of data
Total simulation time	180 sec	Current work
Deadline margin	20% (1.2)	
Buffering time of AEPs	1.0 sec	
DFT_j lambda	1.0 sec	
Request size	320 bytes	[84]

Moreover, each application has an expected deadline to be completed. This deadline is computed as the sum of all the data update times in the application critical path. Thus, the application deadline is the sum of the data update times of requests in the critical path of the application DAG, i.e. the sum of $DUpTm_i$ for data types 1-4, 6-9 and 5, as shown in Table 10, considering a deadline margin shown in Table 11. For generating the data freshness threshold of applications, we used an exponential distribution with a lambda parameter. Choosing a High lambda parameter (>= 1.0) means a denser probability closer to zero, thus we have applications which are more restrictive in their data freshness requirement. When choosing lower lambda (<= 1.0) the applications are more permissive in terms of data freshness threshold. In Section 5.2 we present the evaluation metrics used in our evaluation.

5.2 Metrics

Table 12 summarizes the metrics used in the performed evaluations. TECST is the sum of the energy consumption of the PSANs during simulation. TECET is the sum of the energy consumption of the edge nodes during simulation, for running the VNs and AEPs. The definition of TECCT is similar to the definition of TECET, but for the cloud tier. Considering that, usually, replaceable batteries are the energy source of a PSAN, we infer that a particular PSAN will be the first one to have its battery depleted. This one is the PSAN with the maximum energy consumption among all PSANs in the sensors tier (MECST). Based on that, we define LTST as the time until the first node dies in the sensors tier. We also assume that a common energy source of a PSAN is two AA batteries, capable of providing 16 kJ during the lifetime of the PSAN, resulting in the formula LTST = 16 kJ / (MECST / simulation time).

TABLE 12. METRICS AND ACRONYMS

Metric	Acronym
total energy consumption by the sensors tier	TECST
total energy consumption by the edge tier	TECET
total energy consumption by the cloud tier	TECCT
lifetime of sensors tier	LTST
average makespan of applications	AMSA
percentage of late applications	PLA
percentage of applications completed	PAC
total utility obtained for applications	TUTOB

We also define AMSA, as follows. We consider that the makespan of a single application is computed as the total time, since the application arrival, until its last request is processed by an allocated VN. Then, AMSA is calculated as the average makespan for all applications in simulation. Moreover, some applications may not be completed within the simulation time for two reasons. The first, and less frequent, is that application arrival was randomly set too close to the end of simulation. The second is that the system can be overloaded, and may not be able process all the requests in queues within the simulation time. Therefore, we count the amount of applications completed through PAC. In addition, considering that delay-sensitive applications often have a strict makespan deadline (defined in Section 5.1.3), we define the PLA. The PLA is calculated only among the applications successfully completed. Finally, we define the TUTOB metric as the total utility obtained for applications. Utility is measured in our work for each request as a number between 0 and 100%, and is used in our objective function according to Equation (12). When running a simulation for a given number of requests, the maximum amount of utility that is possible to

achieve is equal to the number of requests times 100%. TUTOB is, therefore, a relative value (between 0% and 100%) that represents the percentage of this maximum amount of utility that was obtained for applications when running a simulation.

5.3 Design and Analysis of Experiments

To execute the four envisioned experiments (E1, E2, E3 and E4), we designed a discrete event simulation using SimPy [81], a process-based discrete-event simulation framework written in standard Python programming language (version 2.7.11) [81]. We used a desktop computer (Intel Core 2 Duo 2.80 GHz processor and 4 GB RAM) to run the simulations in a controlled environment within the Federal University of Rio de Janeiro. Each experiment was performed under 30 repetitions, what provided a reasonable confidence interval of 95% for the results. In this work, we focus on the evaluation of the allocation of VNs to requests in CoS. Therefore, our simulation uses the high-level events to collect the results, rather than implementing all the lower network level details. In this Section, we describe the design of E1, E2 and E3, along with the analysis of the achieved results.

5.3.1 Experiment E1

We designed E1 to assess the scalability of Zeus in terms of the number of VNs and the number of applications, in the most adverse configuration. Such a configuration consists of (i) not sharing the results of requests in common for multiple applications (i.e. not using our UDAG formation algorithm) and (ii) using the EBR and MUR rules as the parameters rule_fcen and rule_fdec, respectively. Considering this configuration, we vary (i) the number of applications and (ii) the number of VNs. Regarding the number of applications, we performed 10 variations, adding 200 new applications in the scenario at each variation. As for the number of VNs, we in fact considered incrementing the number of VNs per data type, one by one and from one to five. When performing such a variation, every new VN added is a replica of an existing VN in the base scenario with nine VNs described in Section 5.1. For instance, we first increment the number of VNs per data type by one, having two VNs per data type and thus a total of 18 VNs. For three VNs per data type, we have 27 VNs in total, and so on. We consider the elastic capacity of the edge, so that every new VN can be instantiated to run on the same host EN of its original replica, and have the same amount of computational resources as its original replica. In addition, new VNs are created along with a new and exclusive set of PSANs

(configured as the other PSANs described in Section 5.1.1), when necessary. Although the physical network density increases when we use this strategy, we consider no interference among underlying PSANs of different VNs. Therefore, we are increasing the capacity of the CoS infrastructure to provide data to an also increasing number of applications. Figure 8 shows the results of E1.

In Figure 8a, the TECST metric achieves the maximum value for 400 applications and one VN per data type (DT). This is the saturation point for one VN per data type. Since we use the MUR rule in E1, the VNs always perform data updates to meet requests. Therefore, 400 applications are enough to minimize the time between every two successive data updates for any given VN. Therefore, for any number of applications greater than the saturation point, the VNs are said to be saturated. For two, three, four and five VNs per DT, the saturation points are, respectively, 800, 1200, 1600 and 2000 applications. In Figure 8b, the LTST metric is higher, as higher is the number of VNs per data type (more than a month for five VNs per data type and 200 applications). The EBR rule balances the load of requests among all available VNs, avoiding overloading the same VN and thus improving the LTST, as the number of VNs per DT increases. In addition, the LTST converges to five days as the number of applications increases, for any amount of VNs per DT. In Figure 8c, the TECET metric increases for processing a number of applications greater than the saturation point, for each amount of VNs per DT. For 2000 applications, the TECET ranges from 6 KJ (one VN per DT) to 10 KJ (five VNs per DT). Since we did not use a cloud node in this experiment, the TECCT is zero.

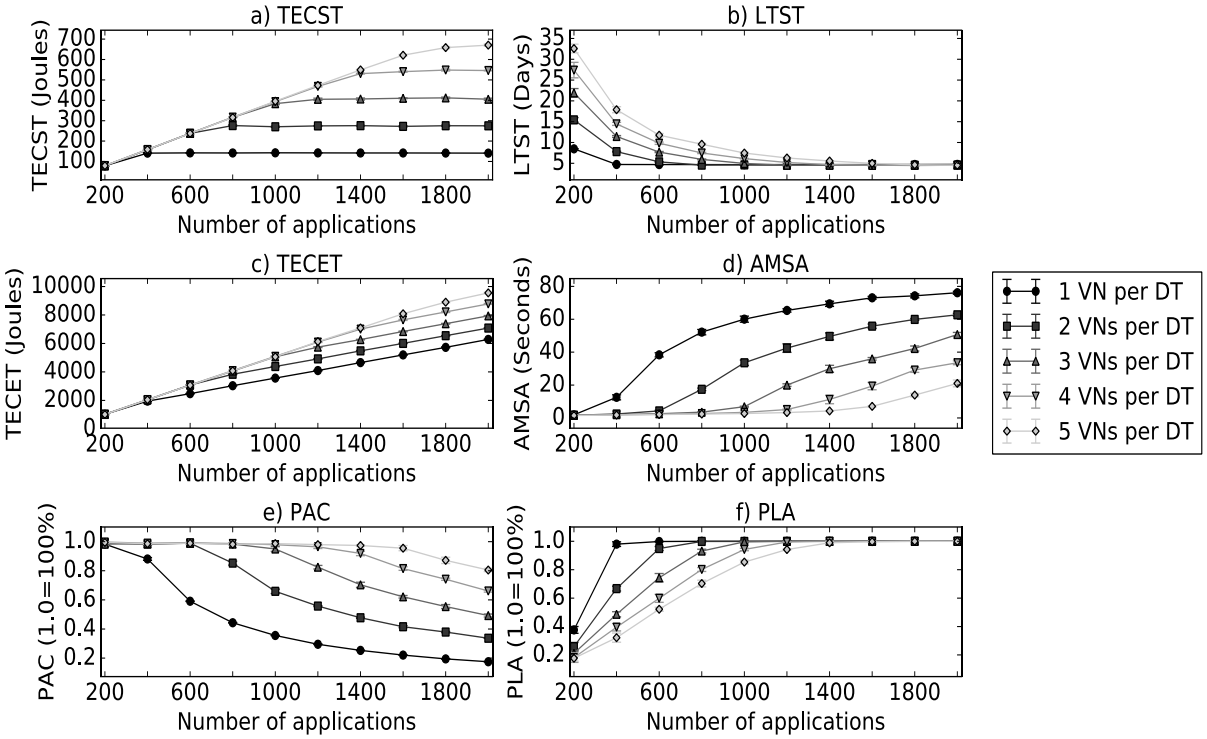

Figure 8. Results of experiment E1

In Figure 8d, the values for AMSA metric grow slower, as the number of VNs per DT increases. The value of AMSA for one VN per DT is close to 1 second for 200 applications, growing to nearly 80 seconds for 2000 applications. This is due to the larger VN queues formed when the number of applications increases. The use of the MUR rule, along with not considering the UDAG formation algorithm, makes VNs always perform a data update for each request individually. This contributes to form such queues, causing delays to applications and increasing their makespan. In addition, when increasing the number of VNs per DT from one to five, the AMSA drops, respectively, from 80 to 20 seconds, for 2000 applications. This is due to the distribution of the load of applications, mainly respective to data updates, through the EBR rule. In Figure 8e, the PAC metric decreases slower as the number of VNs per DT increases. Using five VNs per DT, we achieved the best PAC curve, with a value greater than 80% for 1800 applications. In Figure 8f, the PLA grows slower as the number of VNs per DT increases. For five VNs per DT and 1200 applications, we still find a PLA lower than 100%. Therefore, considering more VNs per DT allows more applications to be completed in simulation time (higher PAC) and within their deadlines (lower PLA).

To prove that our algorithm is scalable in our scenario, we consider the definition of scalability from the area of distributed systems, as follows. Scalability is the capability of a system, network, or process to handle a growing amount of work [102]. In addition, the authors in [93] state that scalability is inherently about concurrency; it is about doing more work at the same time. Therefore, to prove that our algorithm is scalable, in the scenario used in E1 (where the DAG of requests allows the concurrency of requests a priori), we must prove the following statement. When increasing the number of VNs (to process more work/requests), the algorithm can perform more requests (work) simultaneously (concurrently), and without a clear maximum limit in both the number of VNs and requests [93]. In our work, we can prove this by assessing the results obtained for the PAC and PLA metrics. With the increase in the number of VNs per DT, our algorithm can reach higher percentages of applications completed and in time (punctual), without a clear limit. In addition, the saturation point grows linearly as the number of VNs per DT grows. And the values of AMSA grow as a sigmoid curve, as the number of VNs per DT grows. Thus, both metrics do not show an exponential or combinatorial (explosive) growth, which would hinder the scalability of Zeus.

Besides proving that Zeus is scalable, it is important to assess the cost of adding a new VN per DT in Zeus, in order to process growing loads, i.e. proving that such scalability is worth in terms of resources spent to achieve it. Adding a new VN per DT has a positive effect on the LTST, because the workload of requests is better distributed (balanced) among VNs. The LTST raises from around 9 to 33 days when we vary the number of VNs per DT from 1 to 5, respectively. In addition, the TECST and TECET grow linearly, at a constant rate, with the increasing amount of applications, also denoting a non-explosive growth.

Finally, in every aspect assessed in E1, the hybrid approach of Zeus succeeded in handling a growing load (in terms of an increasing number of applications), and in scaling (in terms of an increasing number of VNs) to accommodate such a growing load. Thus, the hybrid approach of Zeus is scalable.

5.3.2 Experiment E2

We designed E2 to assess how the use of the edge tier in Zeus allows supporting delay sensitive applications, in comparison to an approach using the cloud tier only. In E2, we considered four smart buildings (replicating four times the base scenario of Section 5.1). In

addition, we used five VNs per data type, considering the better results achieved for this configuration in E1. However, we consider that each building is independent, having its exclusive set of data types. Moreover, we designed applications, each for one of the four buildings, with the same DAG structure, but with the exclusive data types of the respective building. In addition, we used our UDAG formation algorithm during E2, and considered the QTRR and ANUR rules as the parameters rule_fdec and rule_fcen, respectively. In E2, we vary (i) the use of two distinct scenarios of configurations, one with the edge tier and the other one without it, and (ii) the number of applications. In the configuration using the edge tier, each building has its AEP, which makes local decisions for the building (AEPs in ENs). In the configuration without the edge (AEP in CN), the AEP makes the resource allocation decision for all the four smart buildings together, in the CN. Regarding the number of applications, we performed 10 variations, adding 200 new applications per building (800 in total) in each variation. Figure 9 shows the results of E2.

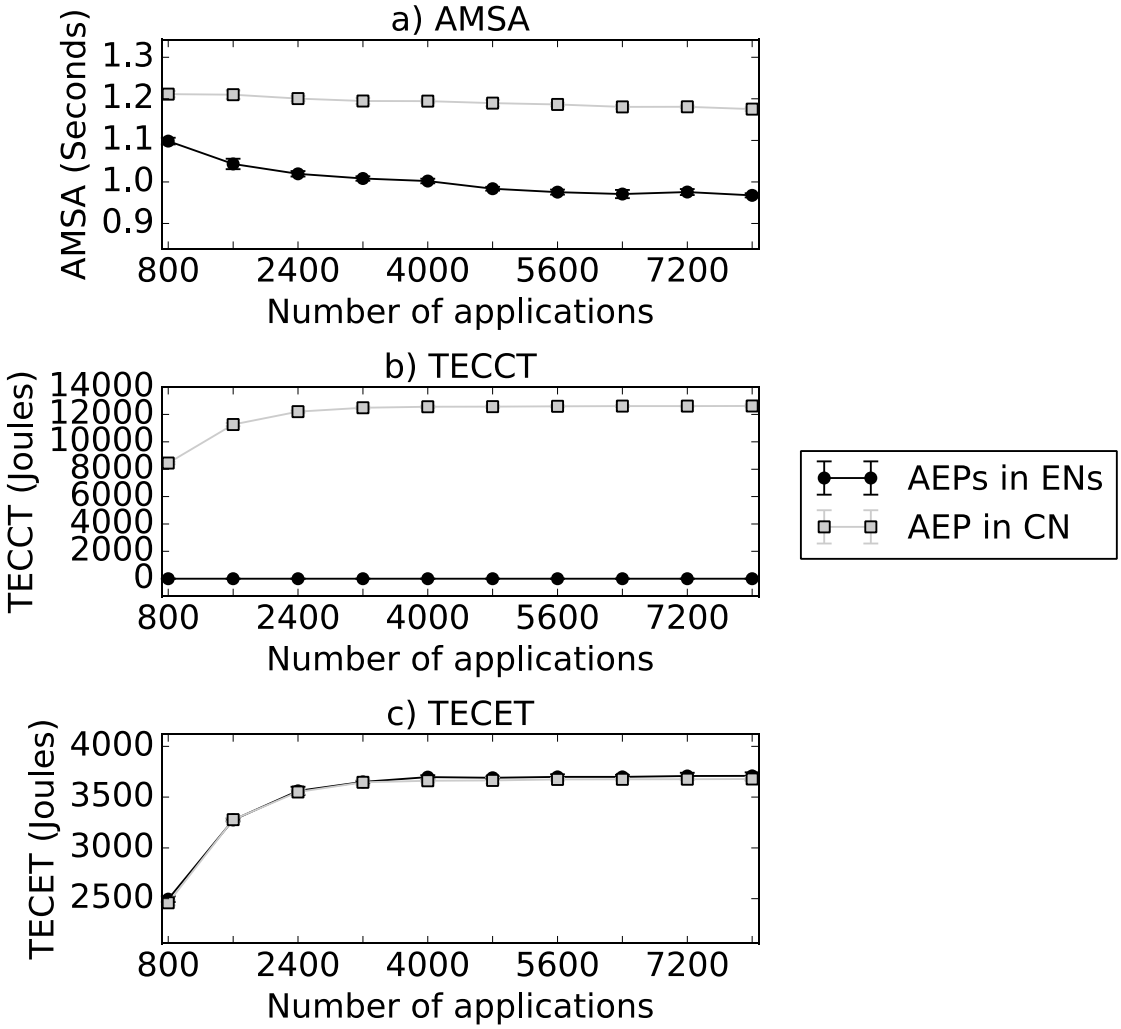

Figure 9. Results of experiment E2

In Figure 9a, the AMSA metric decreases as the number of applications grows, for the configuration with a single AEP in CN. A decreasing behaviour in the values of AMSA is also shown by the configuration of multiple AEPs in ENs. The decreasing behaviour of each curve, separately, occurs because of a higher efficiency achieved by the identification of requests in common for a greater amount of applications, in comparison to using such a mechanism on few applications. Thus, more applications will be able to share the same data, avoiding delays due to data updates, and consequently reducing the makespan of applications, in average. In addition, for 8000 applications, the AMSA is higher (around 1.18 seconds) in the configuration with one AEP in CN, in comparison to the configuration with multiple AEPs in ENs (around 0.97 seconds). We explain this result by the additional time required for communication between the cloud, where the decision is taken, and the edge, which holds VNs. The PAC and PLA values achieved are 100% for every configuration in this experiment. In Figure 9b, the TECCT metric grows to around 12 KJ for 8000 applications, in the configuration with one AEP in CN, and drops to zero when using multiple AEPs in ENs. In Figure 9c, the TECET metric grows to around 3.7 KJ for 8000 applications, in the configuration with one AEP in CN. The TECET is insignificantly greater when using multiple AEPs (less than 1% difference between both curves). This result occurs exclusively due to the need for additional processing in the edge tier, a less time and energy intensive task than communication.

We conclude that leveraging the use of the edge tier is a better option for saving energy and reducing the makespan of applications, because applications take additional time for being communicated between the CT and the ET. The physical proximity of AEPs, VNs and their PSANs is better explored as an intrinsic characteristic of the edge, speeding up communications among such entities in comparison to any approach hosting some of these entities in the cloud. When using the CT to make decisions in the performed experiments, the CN uses an unnecessary amount of energy for processing applications insignificantly faster than the ENs. At some point, the cloud would be more efficient in terms of energy, but only when considering a far more computationally intensive scenario (greater than 8000 applications). Finally, according to Bonomi et al. [39], the delay-sensitive applications require strict response times, in the order of milliseconds to sub seconds (less than one second, which we consider as our baseline). When leveraging the use of the edge tier, we achieved an

application makespan (time for acquiring, deciding and actuating over the acquired data) of less than one second, in average (for 8000 applications). The response time of applications (time for only deciding and actuating over the acquired data) is, consequently, lower than one second, which is a good result in comparison to our baseline. Therefore, Zeus is more suitable for delay sensitive applications, with restrictive response times, when leveraging the use of the edge tier, in comparison to a typical two-tier CoS architecture that faces the delay communication with the cloud, such as [3]. Finally, this is a contribution of our work specifically to improve the area of CoS systems.

5.3.3 Experiment E3

We designed E3 to show how the UDAG formation algorithm used in Zeus contributes for saving resources from the nodes, consequently improving the WSANs lifetime. We used the EBR and MUR rules as the parameters rule_fcen and rule_fdec, respectively. Moreover, we considered the same scenario as E1, however with a fixed number of five VNs per data type, considering the better results achieved for this configuration. In E3 we vary the number of applications using the same method as in E1. In addition, unlike E1, we used two distinct scenarios of configurations, one with the UDAG formation algorithm and the other without it. Figure 10 shows the results of E3.

In Figure 10a, the TECST metric grows linearly until the saturation point of five VNs per data type (2000 applications), when not using the UDAG formation algorithm, as in E1. When using the algorithm, the TECST drops to a nearly constant curve, assuming a value around 90 J for any number of applications. For 2000 applications, TECST drops by 86% when using the algorithm. More importantly, when more applications are used, the possibility of finding common requests among them increases. Thus, the proposed UDAG formation algorithm is able to fully utilize the unique requests to reduce the value of TECST. In Figure 10b, the LTST metric is shown for both configurations (with and without the UDAG formation algorithm). LTST without using the UDAG formation algorithm is the same as in E1. When using the algorithm, LTST is better in comparison to when not using such algorithm, for any given number of applications. For 2000 applications, LTST increases from around 5 to around 38 days when we use the UDAG formation algorithm. The peak lifetime is around 2 months, for 200 applications, when using the UDAG formation algorithm. In Figure 10c, the TECET metric shows curves with shapes similar to the TECST curves, however there is a difference of

proportions between them. The edge tier consumes around 1 KJ for 2000 applications when using the UDAG formation algorithm. The energy saving in the edge tier, when using the UDAG formation algorithm, is explained by the reduced number of requests communicated between the AEPs and VNs, both running on ENs.

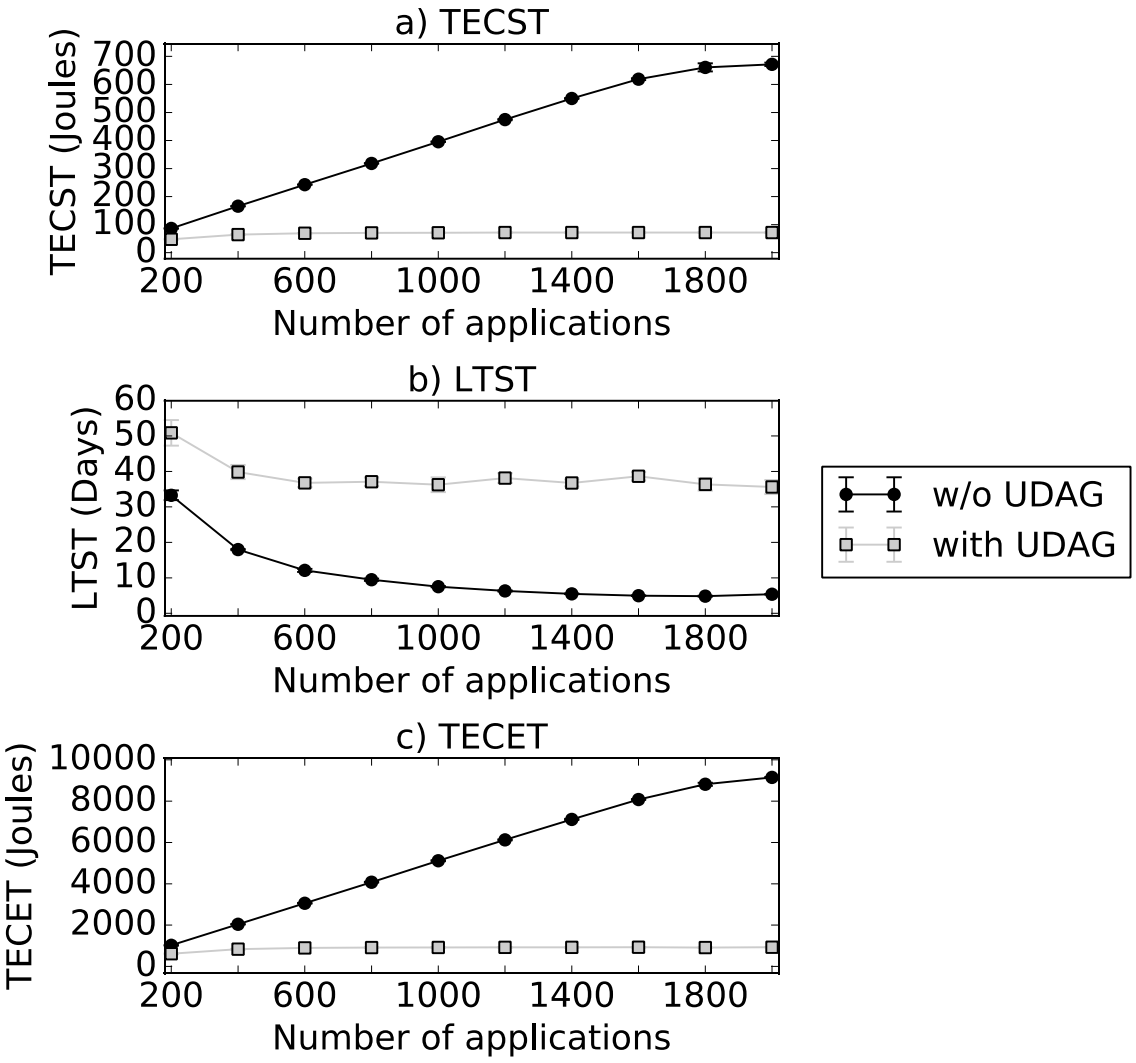

Figure 10. Results of experiment E3

Finally, Zeus allows identifying tasks that are common for multiple applications, performing them only once and sharing the outcome among the applications. Therefore, Zeus reduces the energy consumption of the ST and consequently extends the lifetime of WSANs.

5.3.4 Experiment E4

We designed E4 to assess the quality of the solutions calculated by Zeus, and show how much energy it can save for the WSANs when reusing data among multiple applications.

To assess the quality of solutions provided by Zeus, we used the well-known strategy of comparing the solutions obtained by Zeus with a baseline [99]. It is important to mention that running a traditional optimization algorithm to obtain optimum solutions to use as a baseline takes a prohibitive amount of time and computational resources to run, due to the NP-completeness and combinatorial growth nature of our MINPP. For instance, we implemented a small instance of our problem in the CPLEX [100] software, consisting of 18 VNs (two VNs per DT), 36 requests (4 applications), 4 periods, 45 PSANs (5 PSANs per VN) and 4 edge nodes. The remaining parameters received the same values mentioned in Section 5.1. This small instance took more than one hour running in CPLEX, and could not run to end since it consumed all the RAM memory available in a typical personal laptop configuration (4 GB RAM, Intel Core i5 processor, running windows 10). Therefore, we used a baseline estimated by us, representing an expected optimum solution. Since we know the number of requests in every scenario, and that the maximum utility that can be obtained for each request is 100%, by multiplying both values we can reach a reliable maximum baseline for utility. Moreover, using the MUR as the parameter rule_fdec in Zeus generates the baselines for comparison (utility is maximized with this rule). Therefore, we consider that the implementation of Zeus using the MUR rule is our baseline. We can compare results obtained by Zeus when using the ANUR rule as the parameter rule_fdec to this baseline for utility. Thus, we can have a reliable understanding of how good are the solutions provided by Zeus when considering the reuse of data, sharing such data among multiple applications (respecting their data freshness requirements and reducing utility obtained). The results obtained by using these two rules (MUR and ANUR) are worth comparing, due to their conflicting nature. Besides utility, in E4 we also assessed the energy consumption for achieving each solution, and how much energy can be saved in our comparison with the baseline (MUR). Although the MUR rule maximizes utility, this rule also imposes a greater energy overhead due to data updates on the sensors tier, in comparison to the ANUR rule.

In E4, we used two distinct configurations, one with the MUR rule as the parameter rule_fdec and the other with the ANUR rule. Moreover, we vary the number of applications using the same method as in E1. Besides such variations, we considered as fixed the following parameters: (i) using the UDAG formation algorithm, (ii) using the EBR rule as the parameter rule_fcen and (iii) using five VNs per data type. Figure 11 shows the results of E4.

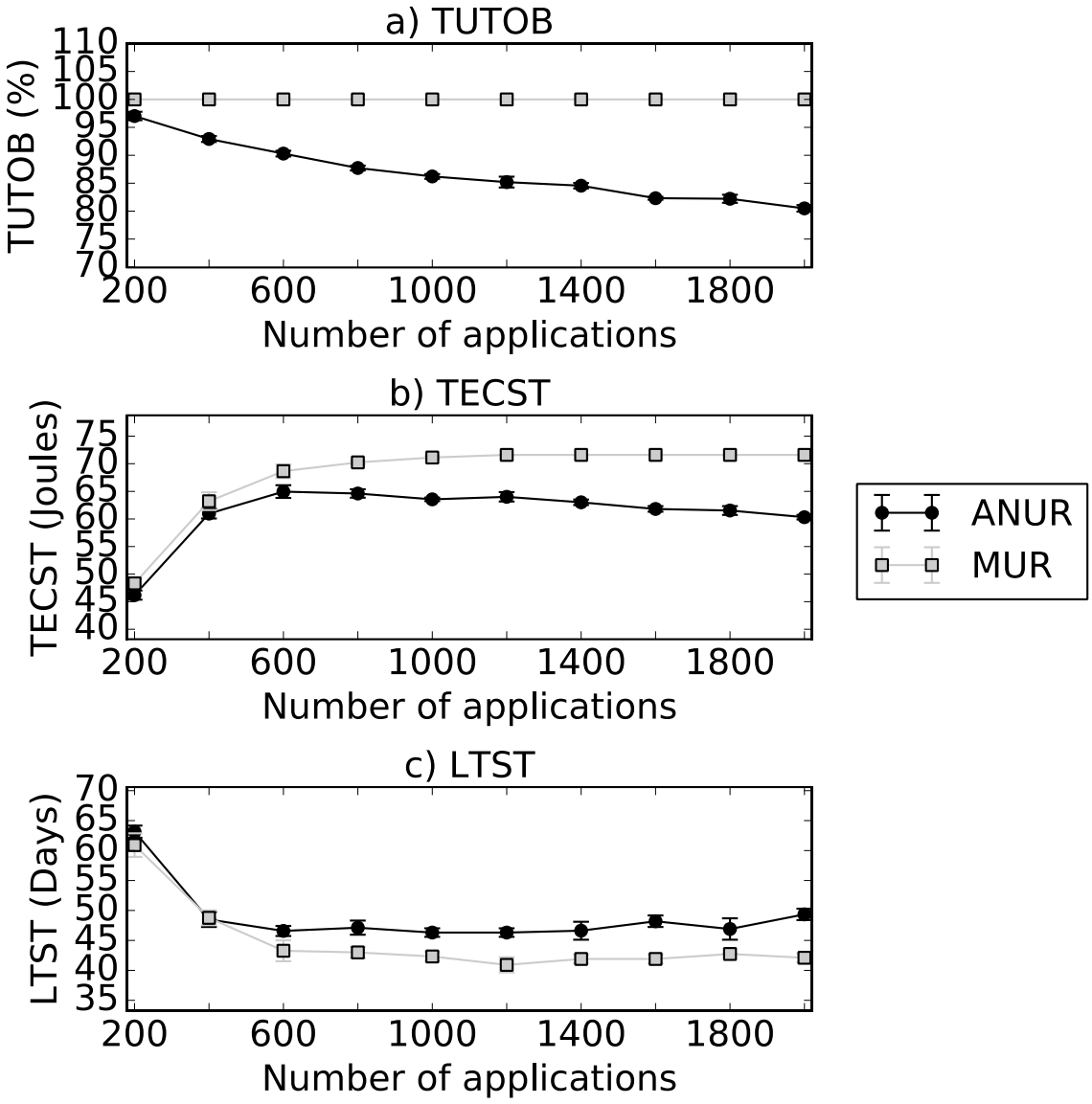

Figure 11. Results of experiment E4

In Figure 11a, the TUTOB metric shows a constant curve at 100% when using the MUR rule, as expected. When using the ANUR rule, the TUTOB metric drops from around 96% for 200 applications to around 80% for 2000 applications. This is because when the number of applications increase, more opportunities of reusing data to meet applications occur. In these opportunities, applications are met with values of the data freshness requirement that are greater than zero, thus obtaining values of utility that are lower than 100%.

In Figure 11b, the TECST metric grows until nearly 71J for 2000 applications when using the MUR, because every VN is performing the maximum number of data updates in the available simulation time. When using the ANUR rule, the TECST is lower in comparison to

when using the MUR. For 2000 applications, the TECST is reduced from 71J to 60J. This represents around 15% of energy saving for the sensors tier.

In Figure 11c, the LTST metric is shown for both configurations (using the MUR and ANUR rules). The LTST is greater when using the ANUR rule in comparison to when using the MUR, as expected, since less energy is spent in the sensors tier using ANUR. For 2000 applications, the LTST raises to around 50 days when using the ANUR rule, in comparison to 42 days, when using the MUR rule.

Finally, the approach of Zeus is based on sharing the data provisioned by VNs among multiple applications, considering the data freshness requirement of applications. This approach is reflected by the algorithm using the ANUR rule. In comparison to the algorithm using the MUR rule, which was used as a baseline, the approach proposed in our work is capable of achieving near-optimal solutions, in the worst case, of values around 80% of the optimal solution. Moreover, by using the ANUR rule, we can save around 15% of energy for the sensors tier, in comparison to the baseline algorithm using the MUR rule.

5.4 On the implementation of the CoS using the FIWARE platform

In this Section we present a brief and initial discussion on how to implement a CoS system based on the set of CoS system components presented in Section 3.3 in an existing platform called FIWARE [104]. The FIWARE project provides an ecosystem that facilitates the development of smart applications in multiple domains. FIWARE is organized in seven technical parts: (i) Cloud Hosting, (ii) Data/Context Management, (iii) IoT Services Enablement, (iv) Security, (v) Applications, Services and Data Delivery, (vi) Interface to Networks and Devices Architecture, and (vii) Advanced Web-based User Interface. To the best of our knowledge, FIWARE is the first initiative that provides all the features required to implement our proposed architecture. Firstly, FIWARE follows the paradigm of integrating smart things to the cloud. Moreover, it introduces an innovative infrastructure composed of public, reusable and generic building blocks, software components called Generic Enablers (GEs). Each GE provides well-defined APIs and interoperable interfaces that comply with the specifications published for that GE, easing application development. GE's open-source reference implementations are publicly available [104]. Furthermore, FIWARE is based on lightweight virtualization, using docker containers to run GEs [104].

In our implementation (Figure 12), the FIWARE GEs described in parts (ii) and (iii) encompass our cloud and edge tiers. However, some GEs are implementations originally centralized at the cloud. To fit the requirements of our architecture, we modified some of these GEs to execute in a decentralized way. Chapter (iii) provides GEs to allow the interaction of FIWARE-based applications with real-world objects. These GEs are spread over two different domains namely IoT Backend and IoT Edge. IoT Backend comprises the GEs hosted in the cloud that provide functionalities such as device managing, device composition and discovery. IoT Edge encompasses infrastructure elements required to connect physical devices to FIWARE applications. It is worth mentioning that we do not implement a cloud. We only depict the typical CoS services that run on the cloud tier. Hence, we discarded chapter (i), whose components are designed to implement a cloud environment. Table 13 describes the GEs that supply the storage and communication infrastructure of the CoS. Our cloud tier is directly related to FIWARE's IoT Backend tier, while FIWARE's IoT Edge is related to our edge and sensor tiers. In the sensors tier, PSANs can implement FIWARE's NGSI9/10 protocol, or a conversion can be performed by the IoT Agent GE (based on HTTP/MQTT), running within a VN. Moreover, to communicate directly with FIWARE's GEs, all the components in our architecture communicate through interfaces that comply with FIWARE's NGSI9/10. In addition, the NGSI Gateway GE implements our EN, and the VS runs on top of it.

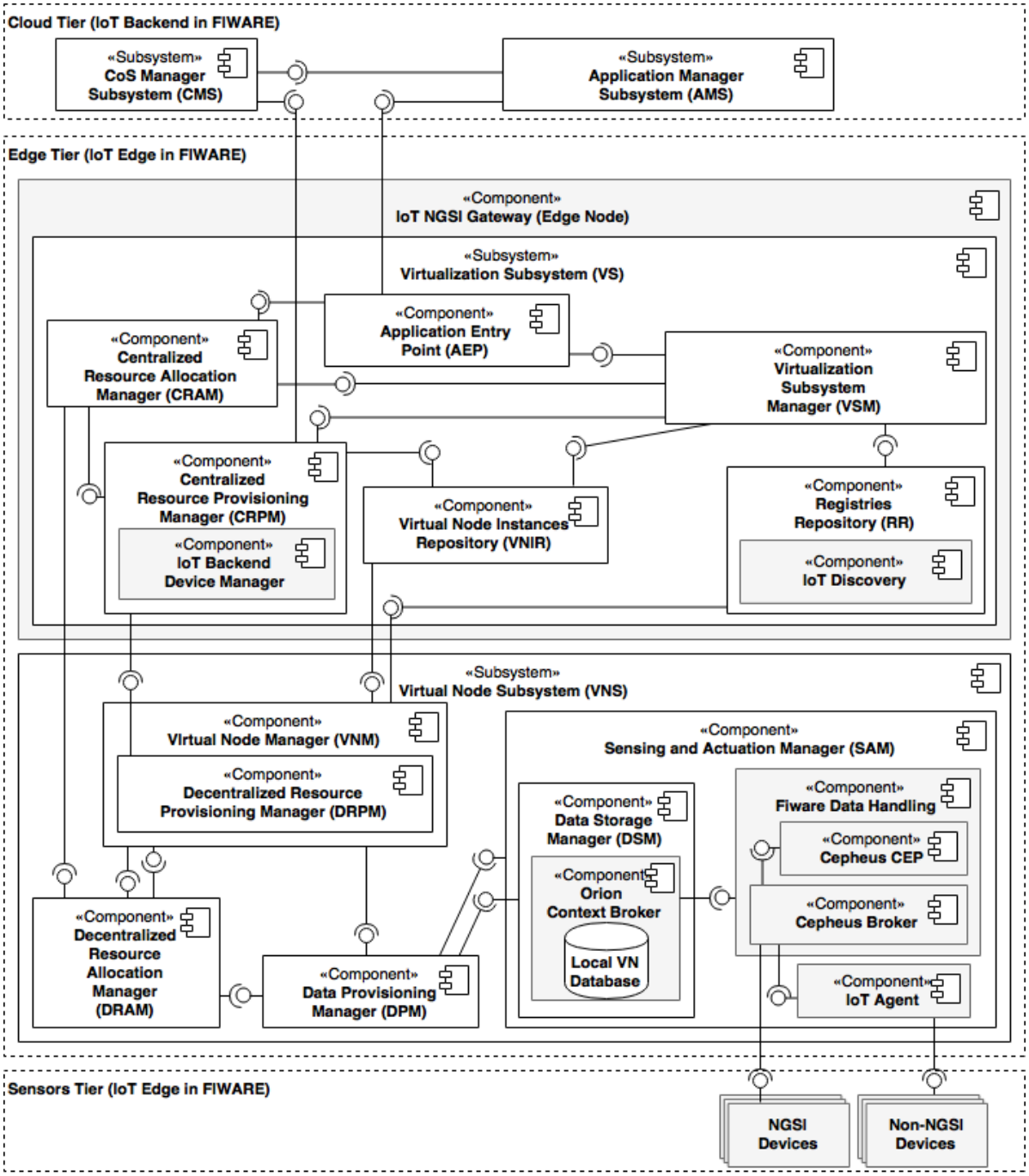

Figure 12. Implementing CoS systems using FIWARE GEs

In the VNS, the SAM component uses the DSM, the Data Handling (DHge) and IoT Agent GEs. The DSM uses the Orion Context Broker GE (CBge), which provides context information (e.g., room temperature, citizen name/sur-name, a bus location) to the DPM, besides performing local storage within a VN using a Mongo database [104]. Besides broker functions, the DSM has an internal structure for implementing the data model used to store historical data, and database queries to retrieve data. The implementation of task scheduling and execution is specific, and thus performed by the SAM with no direct mapping in FIWARE. The

DHge, composed of Cepheus Broker and Cepheus CEP, is used to implement information fusion procedures. The Cepheus Broker is a lightweight broker used for communication between devices and the DSM. The Cepheus CEP is used to process queries on PSANs' data and generate higher-level aggregated events defined by the InPs.

TABLE 13 MAPPING BETWEEN COS COMPONENTS AND FIWARE GES.

CoS Elements	FIWARE GE	Description
PSANs	NGSI9/10, IoT Agent	Communication between the ST and ET. FIWARE NGSI is a RESTFul/HTTP API. NGSI9 is used to exchange information about the availability of context information whereas the purpose of NGSI10 is to exchange context information. The IoT Agent is the software module handling IoT Specific protocols.
Sensing and actuator Manager (SAM)	Data Handling	Addresses the need for filtering, aggregating and merging real-time data from different sources. It includes a Broker and a CEP component.
Data Storage Manager (DSM)	Orion Context Broker	Publish/Subscribe Broker, which allows applications to interchange heterogeneous events following a standard pub– sub paradigm.
Registries Repository (RR)	IoT Discovery	A service discovery mechanism that allows Context Producers (e.g., IoT Agents and Data Handling) to register sensors, actuators, and Things. Also, it enables Context consumers to discover the former registered elements.
Centralized Resource Provisioning Manager (CRPM)	IoT Backend Device Management	Enables creation and configuration of IoT Agents that connect to sensor networks.
-	IoT Broker	It is an IoT Backend enabler that serves as a middleware which enables fast and easy access to information about devices and their attributes.
Application Manager Subsystem (AMS)	-	API for users to request sensing data
CoS Manager Subsystem (CMS)	-	API to manage the CoS system
Decentralized Resource Allocation Manager (DRAM) & Centralized Resource Allocation Manager (CRAM)	-	Execute the orchestration and distribution of data.
Decentralized Resource Provisioning Manager (DRPM)	-	Carries out the decentralized parts of resource provisioning algorithms besides executing the VN boot.

The DRAM/CRAM has the goal of performing the orchestration and distribution of data provided by the DHge, to meet application requests [104], similarly to the IoT Broker's goal. The DRAM/CRAM performs part of these functions, but using a hybrid resource allocation technique, not implemented by any GE.

The CRPM and IoT Backend Device Management GE (BDMge) have the same goals: enabling creation/configuration of IoT Agents that connect to PSANs. The CRPM can use the BDMge in part, since it concentrates the APIs to manage PSANs. However, it is necessary to move the BDMge to run on the edge tier. Similarly, the DRPM could use part of the BDMge in its implementation. However, the GE should be decentralized. Moreover, the DRPM has the goal of performing the VN boot, thus it should be implemented based on docker-compose scripts.

The RR can be implemented using the IoT Discovery GE. However, this GE is originally centralized, and runs in the cloud. It is necessary to decentralize its procedures to run at the edge.

Currently, there is no API in FIWARE to perform CMS' functions, such as creating the data model used within VNs to export data or creating SQLs to perform information fusion within the CEP environment. In the first case, the activity is performed by the InP, using JSON. The created model is then provided to the Context Broker using an HTTP POST. In the second case, an interface provided by the CEP is used. Furthermore, the AMS does not have support on the FIWARE to perform its specific functions.

The remainder components have no direct correlation and support from FIWARE, and should be implemented from scratch. The VNM and VSM provide management functions that are particular of our proposal. The DPM implements a particular service of our architecture. The VNIR hosts VNs, and can be implemented as a data structure in ENs' memory. Finally, the AEP, which can be implemented as a gateway, handling communication between the CT and ET.

Finally, we discussed how FIWARE can support our system implementation, providing the necessary GEs, with little effort for making changes, mainly regarding the decentralization of components. We are currently starting to perform the required changes in FIWARE to generate a full-fledged implementation of our architecture, thus there is no implementation available.

6 Related Work

In this Chapter, we describe relevant works found in literature on resource allocation for the specific research field of CoS. The research on CoS is recent, with most existing proposals focusing on virtualization models, instead of the formulation of resource allocation mechanisms. Thus, in our discussion, we also considered works from the IoT field [54][70][71][73][75][86][87][88], which could be adapted to CoS.

From the field of CoS, Delgado et al. [46] formulated a mixed integer and linear programming problem and proposed a centralized heuristic algorithm based on linear programming to perform resource allocation. Their problem seeks to maximize the number of applications sharing the CoS, while considering constraints related to storage, processing power, bandwidth, and energy consumption requirements of sensors tier. Dinh et al. [49] proposed a centralized model for the CoS to provide on-demand sensing services for multiple applications with generic requirements, such as the data-sensing interval. The authors also designed a request aggregation scheme that considers the more restrictive data-sensing interval among the aggregated requests. Therefore, their scheme performs only once the requests with data sensing intervals in common and shares the results of the consolidated request among the aggregated requests. In addition, the authors formulate a linear problem for minimizing the energy consumption of physical sensors, as well as the bandwidth consumption of sensing traffic.

In the IoT field, Narman et al. [71] propose a generic model, with a greedy algorithm, to perform server allocation to IoT requests dynamically. Moreover, they consider a priority-based queuing of IoT requests. Zeng et al. [73] formulated a mixed integer non-linear programming problem to perform resource allocation in the edge tier. They aim at minimizing the completion time of requests, influenced by computation, I/O interrupts and transmissions, while considering load balancing on both client side and edge side. Moreover, they propose a three-stage algorithm for solving their formulated problem, based on the linear programming relaxation method and a greedy approach. Yu et al. [70] proposed a game-theoretical approach to allocate resources to requests optimally, using virtual machines (VMs) in a cloud-based vehicular network. Their approach is decentralized, such that each VM competes for resources with other VMs based on its local view. The authors aim to meet the QoS

requirements of VMs while ensuring the usage levels of computation and storage resources from the physical infrastructure. Aazam et al. [75], proposed a probabilistic model for resource allocation in the cloud of things using the edge tier to decide what type of data to upload to the cloud tier, avoiding burdening the core network and the cloud. Their approach considers that users and their application requests have an unpredictable probability of stop using resources from the physical infrastructure at any moment. The authors call this a relinquish probability and centre on it the design of their probabilistic model. Moreover, their model also considers as parameters the service type, service price, and variance of the relinquish probability. Farias et al. [54] proposed a framework for Shared Sensor and Actuator Networks (SSAN), including an energy-efficient centralized task scheduling algorithm. A major feature of their work is that the algorithm performs tasks in common to multiple applications only once. Their framework also handles precedencies among requests of the same application when making scheduling decisions. As a major drawback in comparison to our work, their work does not consider the cloud and edge tiers, thus not supporting CoS applications which demand, respectively, high processing capacity for analytics and strict response times. Ekmen et al [88] proposed an energy efficient multi-copy and multi-path WSAN routing strategy. Their basic idea behind multi-copying is to duplicate only the WSAN-generated data that passes through some central nodes, instead of duplicating all the WSAN-generated data, as a precaution against WSAN malfunctioning. A limited number of nodes with higher data transmission rates are determined as central, considering WSAN lifetime maximization objective. As a major strength of this paper, the authors formulate a mixed integer programming problem to determine optimal routing. Moreover, a heuristic algorithm for finding good solutions for large problem instances in reasonable times is proposed. As a drawback, Ekmen et al. do not consider the edge tier in their proposal, with all the potential for supporting WSAN data storage, processing and analytics that such tier can bring, besides the management capabilities and delay-sensitive application support. Xu et al. [86] propose Zenith, a centralized edge computing resource allocation model. Their model allows service providers to establish resource sharing contracts with edge infrastructure providers. Based on the established contracts, service providers employ a latency-aware resource allocation algorithm that enables delay-sensitive requests to run to completion, having their requirements considered in such decision. Their algorithm is auction-based and ensures truthfulness and utility-

maximization for both the Edge Computing Infrastructure Providers and Service Providers. Wang et al. [87] present the Edge NOde Resource Management (ENORM) framework, to address the resource management challenge in the edge tier. In ENORM, provisioning enables cloud servers to offload workloads on to edge nodes. Auto-scaling takes resource availability on the edge node into account and allocates/de-allocates resources provided to a workload. ENORM was able to reduce the latency of applications between 20%-80% and data transfer and communication frequency between the edge node and the cloud by up to 95%. Wang et al. consider the edge tier clearly in their proposal, and their provisioning and auto-scaling mechanisms are designed as centralized solutions. As drawbacks, their work does not consider precedencies among requests of the same application, and requests in common among multiple applications.

Our work differs from all previous proposals mainly because of its hybrid approach. Despite the inherently distributed nature of CoS and IoT, only in [70] a fully decentralized solution is proposed, while all the others are fully centralized approaches. In our hybrid approach, we combine and make the most of the features of both centralized and decentralized solutions. As in decentralized approaches, Zeus is scalable in terms of the number of VNs and applications. As in centralized approaches, each AEP running on the ET has information about a set of local VNs, and centralizes resource allocation decisions for this whole set. Moreover, considering the edge tier in the CoS architecture is another differential of our work, in comparison to [54][46][49], allowing to support delay-sensitive applications.

Our work also differs from all related proposals in the formulation of our optimization problem and solution algorithm. We formulate a MINPP based on the application requirements of data freshness and data type, and our constraints consider the precedencies among requests and the energy consumption of the CoS infrastructure. Thus, Zeus is able to save resources from the CoS infrastructure by reusing data, while improving the data freshness provided to applications. Moreover, heuristic algorithms, such as Zeus, are well known for their low computation overhead. Thus, the computation overhead and time spent for making resource allocation decisions is reduced, allowing the hybrid and online approach of our work.

TABLE 14 SUMMARY OF STRENGTHS OF RELATED WORK AND RESEARCH GAP

	1 Degree of decentralization			2 Handles precedencies	3 Shares requests in common	4 Considers the edge tier
	1.1 Centralized	1.2 Hybrid	1.3 Decentralized			
Farias et al. [54]	X			X	X	
Yu et al. [70]			X			
Narman et al. [71]	X					
Delgado et al. [46]	X					
Zeng et al. [73]	X					X
Dinh et al. [49]	X				X	
Aazam et al. [75]	X					X
Xu et al. [86]	X					X
Wang et al. [87]	X					X
Ekmen et al. [88]	X					
Zeus		X		X	X	X

Moreover, our work differs from [70][71][46][73][49][75][86][87][88] by handling precedencies among requests and from [70][71][46][73][75][86][87][88] by performing requests in common among multiple applications only once, and sharing the result among multiple applications. Dinh et al. [49] also consider requests in common in their solution, although their strategy is different. Our approach for handling requests in common among applications is similar to the one of Farias et al. [54]. Both approaches formulate a unique DAG of requests, rearranging their precedence relationships through a similar algorithm. Our approach differs from both works of Dinh et al. and Farias et al. by considering requests in common based on their negotiable requirement of data freshness, and the non-negotiable requirement of data type. Finally, each one of the aforementioned contributions is not exclusive to our work. Other works also contribute on each feature separately such as considering the edge tier, and supporting delay-sensitive applications. However, no algorithm found in literature supports simultaneously all the four features of, namely, being a hybrid algorithm, handling precedencies among requests, sharing requests in common among multiple applications and considering the edge tier. Such four features are, each, specifically important to the area of CoS for the reasons above-mentioned. Put together, they constitute the research gap that we investigate in our work, which is summarized in Table 14.

7 Final remarks and future directions

In this thesis we tackled challenges posed by the novel paradigm of CoS. The major goals of this thesis regard proposing a novel CoS virtualization model, Olympus, along with a new algorithm to perform resource allocation in CoS, Zeus.

Olympus is a partly decentralized WSAN virtualization model based on information fusion that allows (i) extending physical WSAN lifetime, (ii) reducing response time for applications, and (iii) supporting several centralized and decentralized applications. Olympus is built with the key concept of information fusion because each of the virtual sensors represents the execution of an information fusion technique. This ensures the system to provide data at a given abstraction level of the manipulated data during the information fusion process. Olympus is to be considered a decentralized CoS virtualization model because physical nodes can perform locally the necessary procedures for creating and running the virtual sensor. The virtual sensor creation and operation management is not fully held centrally, within the Cloud. Therefore, in Olympus, the decision processes of applications are performed partly within physical sensors and partly within the Cloud.

Zeus is a hybrid and heuristic-based algorithm to obtain near-optimal solutions in reduced computation time to the MINPP for resource allocation in Clouds of Sensors. The main distinct features of Zeus are as follows: (i) Zeus can perform requests in common for multiple applications only once, sharing the results of this single execution among these multiple applications. (ii) Zeus considers the existence of precedence relationships (dependencies between data inputs and outputs) among the requests of a same application. (iii) Zeus leverages the concept of edge computing in its operation. (iv) Zeus has characteristics of both centralized and decentralized algorithms. Therefore, the partly-decentralized design of Zeus makes the most of the features of each computational tier of the CoS system.

The key features of Zeus inherit the key features of Olympus, and both provide the major contributions of our approach. As our first contribution, Zeus is scalable, both in terms of the number of VNs and the number of applications executed in the CoS. This contribution is a consequence of the hybrid approach of Zeus, which would not be possible without the support of a decentralized virtualization model as Olympus. Olympus contributes for enabling the decentralization of the WSAN virtualization process, and by supporting both centralized

and decentralized applications simultaneously within the CoS infrastructure. We assessed the scalability of Zeus in the most adverse configuration. In the performed evaluation, the energy consumption of the sensors and edge tiers grow linearly with the increasing amount of applications (ranging from 200 to 2000) running in the CoS system. Moreover, the values of makespan of applications grow as a sigmoid curve, as the number of VNs per data type grows (from 1 to 5). Therefore, in every aspect assessed in our experiments, Zeus succeeded in handling a growing load (in terms of an increasing number of applications), and in scaling (in terms of an increasing number of VNs) to accommodate such a growing load. Thus, the hybrid approach of Zeus achieved a good scalability.

As our second contribution, Zeus provides support to delay-sensitive applications. This contribution is achieved by leveraging edge tier, what allows supporting delay-sensitive applications (three-tier approach), in comparison to an approach using only the cloud tier (two-tier approach). Once again, Olympus plays a key role to achieve this contribution, since it considers the edge tier, allowing the design of Zeus for the three-tier CoS. This contribution is also a result of the use of both the use of information fusion and the decentralization of the WSAN virtualization process considered by Olympus, which reduce the response time of applications, by reducing communication overhead in CoS. Moreover, Zeus is capable of finding solutions in reduced computation time to the problem of resource allocation in CoS, also contributing to support delay-sensitive applications. In the performed experiments, Zeus achieved an application makespan (time for acquiring, deciding and actuating over the acquired data) of less than one second, in average (for 8000 applications). The response time of applications (time for only deciding and actuating over the acquired data) is, consequently, lower than one second. Therefore, Zeus is more suitable for delay sensitive applications, with restrictive response times, when leveraging the use of the edge tier, in comparison to a typical two-tier CoS architecture.

As our third contribution, Zeus saves energy and consequently improves the lifetime of WSANs. This contribution is a direct consequence of using Zeus mechanism for identifying tasks that are common for multiple applications, performing them only once and sharing the outcome among these multiple applications. When using the UDAG formation algorithm, and for 2000 applications running in the CoS, the energy consumption of the sensors tier is reduced by 86%. Also, for 2000 applications, the lifetime of the sensors tier increases from

around 5 to around 38 days when we use the UDAG formation algorithm. Consequently, Zeus contributes for saving energy and extends the lifetime of WSANs. It is important to mention that this contribution would not be possible without considering the key features of Olympus that also contribute to the improvement of the lifetime of WSANs. In this case, the key features are the use of information fusion techniques for reducing data transmission, and the decentralization of WSAN virtualization procedures. We also explicited the trade-off between the quality of solutions found by Zeus and the respective amount of energy consumed by WSANs to achieve them. We showed how much energy can be saved for the WSANs when adopting the approach of reusing data obtained in different moments in time to meet applications, while considering their negotiable data freshness requirements. This approach is reflected by the algorithm using the ANUR rule. In comparison to the algorithm using the MUR rule, which was used as a baseline, the approach proposed in our work is capable of achieving near-optimal solutions, in the worst case, of values around 80% of the optimal solution. Moreover, by using the ANUR rule, we can save around 15% of energy for the sensors tier, in comparison to the baseline algorithm using the MUR rule.

7.1 Future work

There are several research directions that deserve further investigation. One of them is proposing new rules for Zeus, so that its behaviour can be tailored to different QoS requirements and/or application domains. Designing a reactive resource provisioning mechanism, and comparing its performance with the proactive strategy used in Zeus is another avenue we intend to pursue. We also aim at providing support to event-based applications. In addition, we suggest investigating different types of heuristics for solving the MINPP of resource allocation. Such new heuristics could be compared to Zeus using the MUR and ANUR rules, in terms of the quality of solutions obtained, using the same baseline of our current work. Finally, designing a fully autonomous and self-adaptive algorithm, which operates based on a set of rules and policies, and whose adaptation is based on learning techniques, is a final and ambitious goal that we believe will contribute to the fully realization of the potentials of a 3-tier CoS architecture.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[2] A. Botta, W. de Donato, V. Persico, and A. Pescape, "On the Integration of Cloud Computing and Internet of Things," in *2014 International Conference on Future Internet of Things and Cloud*, 2014, pp. 23–30..

[3] S. Madria, V. Kumar, and R. Dalvi, "Sensor Cloud: A Cloud of Virtual Sensors," *IEEE Softw.*, vol. 31, no. 2, pp. 70–77, Mar. 2014.

[4] D. Phan, J. Suzuki, S. Omura, and K. Oba, "Toward Sensor-Cloud Integration as a Service: Optimizing Three-tier Communication in Cloud-integrated Sensor Networks," in *Proceedings of the 8th International Conference on Body Area Networks*, 2013, vol. 1.

[5] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: early architecture and research perspectives," *IEEE Netw.*, vol. 29, no. 3, pp. 104–112, May 2015.

[6] A. Alamri, W. S. Ansari, M. M. Hassan, M. S. Hossain, A. Alelaiwi, and M. A. Hossain, "A Survey on Sensor-Cloud: Architecture, Applications, and Approaches," *Int. J. Distrib. Sens. Networks*, vol. 2013, pp. 1–18, 2013.

[7] M. M. Islam, M. M. Hassan, G.-W. Lee, and E.-N. Huh, "A Survey on Virtualization of Wireless Sensor Networks," *Sensors*, vol. 12, no. 12, pp. 2175–2207, Feb. 2012.

[8] K. Kim, S. Lee, H. Yoo, and D. Kim, "Agriculture Sensor-Cloud Infrastructure and Routing Protocol in the Physical Sensor Network Layer," *Int. J. Distrib. Sens. Networks*, vol. 2014, pp. 1–12, 2014.

[9] E. F. Nakamura, A. a. F. Loureiro, and A. C. Frery, "Information fusion for wireless sensor networks," *ACM Comput. Surv.*, vol. 39, no. 3, p. 9–es, Sep. 2007.

[10] S. M. Ajmal, S. Paris, Z. Zhang, and F. N. Abdesselam, "An Efficient Admission Control Algorithm for Virtual Sensor Networks," *2014 IEEE Intl Conf High Perform. Comput. Commun. 2014 IEEE 6th Intl Symp Cybersp. Saf. Secur. 2014 IEEE 11th Intl Conf Embed. Softw. Syst*, pp. 735–742, 2014.

[11] I. L. dos Santos, L. Pirmez, É. T. Lemos, F. C. Delicato, L. A. Vaz Pinto, J. N. de Souza, and A. Y. Zomaya, "A localized algorithm for Structural Health Monitoring using wireless sensor networks," *Inf. Fusion*, vol. 15, pp. 114–129, Jan. 2014.

[12] W. Li, I. Santos, F. C. Delicato, P. F. Pires, L. Pirmez, W. Wei, H. Song, A. Zomaya, and S. Khan, "System modelling and performance evaluation of a three-tier Cloud of Things," *Futur. Gener. Comput. Syst.*, vol. 70, pp. 104–125, May 2017.

[13] M. Bouzeghoub, "A framework for analysis of data freshness," *Proc. 2004 Int. Work. Inf. Qual. informational Syst. - IQIS '04*, p. 59, 2004.

[14] P. Missier, and S. Embury, "Provider issues in quality-constrained data provisioning," In *Proceedings of the 2nd international workshop on Information quality in information systems (IQIS '05)*. ACM, New York, NY, USA, 2005, 5-15.

[15] C. Perera, A. Zaslavsky, C. H. Liu, M. Compton, P. Christen, and D. Georgakopoulos, "Sensor Search Techniques for Sensing as a Service Architecture for the Internet of Things," *IEEE Sens. J.*, vol. 14, no. 2, pp. 406–420, Feb. 2014.

[16] F. Costa, "Aplicações de técnicas de otimização a problemas de planejamento operacional de lavra em minas a céu aberto," *Dissertação de mestrado*, PPGEM / UFOP, Ouro Preto, M.G, 2005.

[17] P. Raghavendra, "Approximating NP-hard Problems Efficient Algorithms and their Limits,"

2009.

[18]F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog Computing: A Platform for Internet of Things and Analytics," in: *Big Data and Internet of Things: A Roadmap for Smart Environments*, N. Bessis, C. Dobre (Eds.), Springer International Publishing, 2014, pp. 169-186

[19]M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing," *Network-Based Inf. Syst.*, pp. 1–8, 2010.

[20]Open Geospatial Consortium. [Online]. Available: http://www. opengeospatial.org/

[21]B. Rao, P. Saluia, N. Sharma, A. Mittal, and S.V. Sharma, "Cloudcomputing for Internet of Things & sensing based applications," In *Sensing Technology (ICST), 2012 Sixth International Conference on*, pages 374–380. IEEE, 2012.

[22]I. L. Santos, L. Pirmez, L. R. Carmo, P. F. Pires, F. C. Delicato, S. U. Khan, and A. Y. Zomaya, "A Decentralized Damage Detection System for Wireless Sensor and Actuator Networks," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1363–1376, May 2016.

[23]C. Wu, Y. Xu, Y. Chen, and C. Lu, "Submodular game for distributed application allocation in shared sensor networks," *Proc. - IEEE INFOCOM*, pp. 127–135, 2012.

[24]W. Li, F. C. Delicato, P. F. Pires, Y. C. Lee, A. Y. Zomaya, C. Miceli, and L. Pirmez, "Efficient allocation of resources in multiple heterogeneous Wireless Sensor Networks," *J. Parallel Distrib. Comput.*, vol. 74, no. 1, pp. 1775–1788, Jan. 2014.

[25]H. Liu, A. Nayak, and I. Stojmenovic, "Applications, models, problems, and solution strategies," In *Wireless Sensor and Actuator Networks*, chapter 1, pages 4–7. John Wiley Sons, Inc., Hoboken, New Jersey, 2010.

[26]P. Rawat, K. D. Singh, H. Chaouchi, and J. M. Bonnin, "Wireless sensor networks: a survey on recent developments and potential synergies," *J. Supercomput.*, vol. 68, no. 1, pp. 1–48, Oct. 2013.

[27]M. Aazam, and E. N. Huh, "Fog Computing and Smart Gateway Based Communication for Cloud of Things," *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, Barcelona, 2014, pp. 464-470.

[28]M. Iman, F. C. Delicato, C. M. de Farias, L. Pirmez, I. L. dos Santos, and P. F. Pires, "THESEUS: A Routing System for Shared Sensor Networks," *2015 IEEE Int. Conf. Comput. Inf. Technol. Ubiquitous Comput. Commun. Dependable, Auton. Secur. Comput. Pervasive Intell. Comput.*, vol. 6, pp. 108–115, 2015.

[29]R. Eltarras and M. Eltoweissy, "Adaptive multi-criteria routing for shared sensor-actuator networks," *GLOBECOM - IEEE Glob. Telecommun. Conf.*, 2010.

[30]A. Chirichiello, "Two Formal Approaches for Web Services : Process Algebras & Action Languages," Sapienza Universita di Roma, 2008.

[31]D. Ganesarajah and E. Lupu, "Workflow-based composition of WEB-services: A business model or a programming paradigm?," *Proc. - 6th Int. Enterp. Distrib. Object Comput. Conf.*, vol. 2002–Janua, no. January, pp. 273–284, 2002.

[32]D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE Trans. Services Comput.*, vol. 3, no. 3, pp. 223–235, Jul./Sep. 2010.

[33]R. Liu and I. J. Wassell, "Opportunities and challenges of wireless sensor networks using cloud services," in *Proceedings of the workshop on Internet of Things and Service Platforms - IoTSP '11*, 2011, pp. 1–7.

[34]S. Aleksic, "Green ICT for sustainability: A holistic approach," *2014 37th Int. Conv. Inf. Commun. Technol. Electron. Microelectron.*, no. May, pp. 426–431, May 2014..

[35]A. Hopper and A. Rice, "Computing for the future of the planet," *Philos. Trans. R. Soc. A*

Math. Phys. Eng. Sci., vol. 366, no. 1881, pp. 3685–3697, Oct. 2008.

[36]S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing," in *Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata '15*, 2015, pp. 37–42.

[37]D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3702–3712, Dec. 2016.

[38]M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," in *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14-23, Oct.-Dec. 2009.

[39]F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "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*, 2012, p. 13.

[40]I. L. Santos, L. Pirmez, F. C. Delicato, S. U. Khan, and A. Y. Zomaya, "Olympus: The Cloud of Sensors," *IEEE Cloud Comput.*, vol. 2, no. 2, pp. 48–56, Mar. 2015.

[41]G. E. Gonçalves, P. T. Endo, T. D. Cordeiro, D. Palhares, André Vitor de Almeida Sadok, J. Kelner, B. Melander, and J.-E. Mångs, "Resource allocation in clouds: concepts, tools and research challenges," *Minicursos - XXIX Simpósio Bras. Redes Comput. e Sist. Distrib.*, pp. 197–240, 2011.

[42]B. Gonçalves, J. G. P. Filho, and G. Guizzardi, "A service architecture for sensor data provisioning for context-aware mobile applications," *Proc. 2008 ACM Symp. Appl. Comput. - SAC '08*, p. 1946, 2008.

[43]R. Yang, T. Wo, C. Hu, J. Xu, and M. Zhang, "D^2PS: A Dependable Data Provisioning Service in Multi-tenant Cloud Environment," *2016 IEEE 17th Int. Symp. High Assur. Syst. Eng.*, pp. 252–259, 2016.

[44]M. Bauer, N. Bui, C. Jardak, and A. Nettsträter, "The IoT ARM Reference Manual," in *Enabling Things to Talk*, A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. van Kranenburg, S. Lange, and S. Meissner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 213–236.

[45]C. M. De Farias, W. Li, F. C. Delicato, L. Pirmez, A. Y. Zomaya, P. F. Pires, and J. N. De Souza, "A Systematic Review of Shared Sensor Networks," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–50, Feb. 2016.

[46]C. Delgado, J. R. Gállego, M. Canales, J. Ortín, S. Bousnina, and M. Cesana, "On optimal resource allocation in virtual sensor networks," Ad Hoc Networks, vol. 50, pp. 23–40, Nov. 2016.

[47]D. Young, "Multiobjective Optimization". In *Computational Drug Design*, Vol. 1, 2009,.pp. 237–240. Hoboken, NJ, USA: John Wiley & Sons.

[48]F. Zhang, J. Cao, K. Li, S. U. Khan, and K. Hwang, "Multi-objective scheduling of many tasks in cloud platforms," *Futur. Gener. Comput. Syst.*, vol. 37, pp. 309–320, Jul. 2014.

[49]T. Dinh and Y. Kim, "An Efficient Interactive Model for On-Demand Sensing-As-A-Services of Sensor-Cloud," *Sensors*, vol. 16, no. 7, p. 992, Jun. 2016.

[50]L. Fuentes and D. Jiménez, "An aspect-oriented ambient intelligence middleware platform," in *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing - MPAC '05*, 2005, no. 2, pp. 1–8.

[51]W. Li, F. C. Delicato, and A. Y. Zomaya, "Adaptive energy-efficient scheduling for hierarchical wireless sensor networks," *ACM Trans. Sens. Networks*, vol. 9, no. 3, pp. 1–34, May 2013.

[52]S. Chatterjee and S. Misra, "Optimal composition of a virtual sensor for efficient virtualization within sensor-cloud," *IEEE Int. Conf. Commun.*, vol. 2015–Septe, pp. 448–453, 2015..

[53]M. R. Garey and D. S. Johnson, *Computers and intractability*. 1979.
[54]C. M. de Farias, L. Pirmez, F. C. Delicato, W. Li, A. Y. Zomaya, and J. N. de Souza, "A scheduling algorithm for shared sensor and actuator networks," *Int. Conf. Inf. Netw. 2013*, pp. 648–653, 2013.
[55]W. Li, F. C. Delicato, P. F. Pires, Y. C. Lee, A. Y. Zomaya, C. Miceli, and L. Pirmez, "Efficient allocation of resources in multiple heterogeneous Wireless Sensor Networks," *J. Parallel Distrib. Comput.*, vol. 74, no. 1, pp. 1775–1788, Jan. 2014.
[56]H. Rowaihy, M. P. Johnson, O. Liu, A. Bar-Noy, T. Brown, and T. La Porta, "Sensor-mission assignment in wireless sensor networks," *ACM Trans. Sens. Networks*, vol. 6, no. 4, pp. 1–33, Jul. 2010.
[57]J. Barbaran, M. Diaz, and B. Rubio, "A Virtual Channel-Based Framework for the Integration of Wireless Sensor Networks in the Cloud," in *2014 International Conference on Future Internet of Things and Cloud*, 2014, pp. 334–339.
[58]S. Saha, "Secure sensor data management model in a sensor - cloud integration environment," in *2015 Applications and Innovations in Mobile Computing (AIMoC)*, 2015, pp. 158–163.
[59]C. Das and S. P. Tripathy, "A Review on Virtualization in Wireless Sensor Network," vol. 1, no. 1, pp. 1–8, 2010.
[60]I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: A survey," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 553–576, 2016.
[61]M. M. Islam, M. M. Hassan, G.-W. Lee, and E.-N. Huh, "A Survey on Virtualization of Wireless Sensor Networks," *Sensors*, vol. 12, no. 12, pp. 2175–2207, Feb. 2012.
[62]G. Hackmann, F. Sun, N. Castaneda, C. Lu, and S. Dyke, "A holistic approach to decentralized structural damage localization using wireless sensor networks," *Comput. Commun.*, vol. (In Press), p. 13, Jan. 2012.
[63]O. Sinnen, "Wiley Series on Parallel and Distributed Computing," in *Task Scheduling for Parallel Systems*, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2007, pp. 297–298.
[64]P. Raghavendra, "Approximating NP-hard Problems Efficient Algorithms and their Limits," 2009.
[65]M. R. Garey and D. S. Johnson, *Computers and intractability*. 1979.
[66]R. I. Bolaños, M. G. Echeverry, and J. W. Escobar, "A multiobjective non-dominated sorting genetic algorithm (NSGA-II) for the Multiple Traveling Salesman Problem," *Decis. Sci. Lett.*, vol. 4, no. 4, pp. 559–568, 2015.
[67]R. J. Vanderbei, "Linear Programming: Foundations and Extensions," *J. Oper. Res. Soc.*, vol. 49, no. 1, pp. 94–94, Jan. 1998.
[68]J. Zhang, H. Huang, and X. Wang, "Resource provision algorithms in cloud computing: A survey," *J. Netw. Comput. Appl.*, vol. 64, pp. 23–42, Apr. 2016.
[69]C. M. de Farias, L. Pirmez, F. C. Delicato, W. Li, A. Y. Zomaya, and J. N. de Souza, "A scheduling algorithm for shared sensor and actuator networks," *Int. Conf. Inf. Netw. 2013*, pp. 648–653, 2013
[70]R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang, "Toward cloud-based vehicular networks with efficient resource management," *IEEE Netw.*, vol. 27, no. 5, pp. 48–55, Sep. 2013.
[71]H. S. Narman, M. S. Hossain, M. Atiquzzaman, and H. Shen, "Scheduling internet of things applications in cloud computing," *Ann. Telecommun.*, 2016.
[72]C. Delgado, J. R. Gállego, M. Canales, J. Ortín, S. Bousnina, and M. Cesana, "On optimal resource allocation in virtual sensor networks," Ad Hoc Networks, vol. 50, pp. 23–40, Nov. 2016.

[73]D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, “Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System,” *IEEE Trans. Comput.*, vol. PP, no. 99, pp. 1–1, 2016.

[74]T. Dinh and Y. Kim, “An Efficient Interactive Model for On-Demand Sensing-As-A-Services of Sensor-Cloud,” *Sensors*, vol. 16, no. 7, p. 992, Jun. 2016.

[75]M. Aazam, M. St-Hilaire, C.-H. Lung, and I. Lambadaris, “PRE-Fog: IoT trace based probabilistic resource estimation at Fog,” in *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2016, pp. 12–17.

[76]Y. Zhang and J. Chen, “Constructing scalable Internet of Things services based on their event-driven models,” *Concurr. Comput. Pract. Exp.*, vol. 27, no. 17, pp. 4819–4851, Dec. 2015.

[77]I. C. Wong, Z. Shen, B. L. Evans, and J. G. Andrews, “A Low Complexity Algorithm for Proportional Resource Allocation in OFDMA Systems,” in *IEEE Workshop onSignal Processing Systems, 2004. SIPS 2004.*, 2004, pp. 1–6.

[78]C. H. Papadimitriou, “Computational complexity,” Reading, Mass. [u.a.]: Addison-Wesley, 2005.

[79]D. G. Luenberger, G. David, Y. Ye, “Linear and nonlinear programming,” International Series in Operations Research & Management Science. 116 (Third ed.). New York: Springer. pp. xiv+546.

[80]J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, “MICAz DataSheet,” MEMSIC, USA Available: http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf, 2010.

[81]N. Matloff, “Introduction to Discrete-Event Simulation,” in *Introduction to Discrete Event Systems*, Boston, MA: Springer US, 2008, pp. 557–615.

[82]J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, “System architecture directions for networked sensors,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 34, no. 5, pp. 93–104, Dec. 2000.

[83]D.Moss, and P. Levis, “BoX-MACs: Exploiting physical and link layer boundaries in low-power networking,” *Computer Systems Laboratory*, Stanford University, Stanford, CA, USA, Tech. Rep. SING-08-00, 2008.

[84]B. A. Mah, “An empirical model of HTTP network traffic,” in *Proceedings of INFOCOM ’97*, 1997, vol. 2, no. April 1995, pp. 592–600.

[85]R. Martí and G. Reinelt, “Heuristic Methods,” in *European Journal of Operational Research*, vol. 241, no. 3, 2011, pp. 17–40.

[86]J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, “Zenith: Utility-Aware Resource Allocation for Edge Computing,” in *2017 IEEE International Conference on Edge Computing (EDGE)*, 2017, no. June, pp. 47–54.

[87]N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos, “ENORM: A Framework For Edge NOde Resource Management,” *IEEE Trans. Serv. Comput.*, vol. 46, no. November, pp. 1–1, 2017.

[88]M. Ekmen and A. Altın-Kayhan, “Reliable and energy efficient wireless sensor network design via conditional multi-copying for multiple central nodes,” *Comput. Networks*, vol. 126, pp. 57–68, Oct. 2017.

[89]M. Arghavani, M. Esmaeili, M. Esmaeili, F. Mohseni, and A. Arghavani, “Optimal energy aware clustering in circular wireless sensor networks,” *Ad Hoc Networks*, vol. 65, pp. 91–98, Oct. 2017.

[90]I. Ez-zazi, M. Arioua, and A. El Oualkadi, “On the design of coding framework for energy efficient and reliable multi-hop sensor networks,” *Procedia Comput. Sci.*, vol. 109, pp. 537–544, 2017.

[91]A. Laouid, A. Dahmani, A. Bounceur, R. Euler, F. Lalem, and A. Tari, “A distributed multi-path routing algorithm to balance energy consumption in wireless sensor networks,” *Ad Hoc Networks*, vol. 64, pp. 53–64, Sep. 2017.

[92]K. Tang, R. Shi, and J. Dong, “Throughput analysis of cognitive wireless acoustic sensor networks with energy harvesting,” *Futur. Gener. Comput. Syst.*, p. , Aug. 2017.

[93]P. Li and S. Zdancewic, “Combining events and threads for scalable network services implementation and evaluation of monadic, application-level concurrency primitives,” *ACM SIGPLAN Not.*, vol. 42, no. 6, p. 189, Jun. 2007.

[94]D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, “Resource pool management: Reactive versus proactive or let’s be friends,” *Comput. Networks*, vol. 53, no. 17, pp. 2905–2922, Dec. 2009.

[95]N. Kaur Kapoor, S. Majumdar, and B. Nandy, “Techniques for Allocation of Sensors in Shared Wireless Sensor Networks,” *J. Networks*, vol. 10, no. 1, pp. 15–28, 2015.

[96]L. Sharifi, N. Rameshan, F. Freitag, and L. Veiga, “Energy Efficiency Dilemma: P2P-cloud vs. Datacenter,” in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, 2014, vol. 2015–Febru, no. February, pp. 611–619.

[97]I.L. Santos, F.C. Delicato, L. Pirmez, P.F. Pires, A. Y. Zomaya, “Resource allocation and task scheduling in the cloud of sensors”, in: *The Philosophy of Mission-Oriented Wireless Sensor Networks*, eds. Habib Ammari, Springer, 2017 (Accepted for Publication).

[98]R. de Paz Alberola and D. Pesch, “AvroraZ,” in *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks - PM2HW2N ’08*, 2008, pp. 43–50.

[99]R. Martí and G. Reinelt, “Heuristic Methods,” in *European Journal of Operational Research*, vol. 241, no. 3, 2011, pp. 17–40.

[100] CPLEX. CPLEX Manual, 1998. URL http://www.cplex.com.

[101] P. Patel and D. Cassou, “Enabling high-level application development for the Internet of Things,” *J. Syst. Softw.*, vol. 103, no. C, pp. 62–84, May 2015.

[102] A. B. Bondi, “Characteristics of scalability and their impact on performance,” in *Proceedings of the second international workshop on Software and performance - WOSP ’00*, 2000, pp. 195–203.

[103] K. Nishant, P. Sharma, V. Krishna, C. Gupta, K. P. Singh, Nitin, and R. Rastogi, “Load Balancing of Nodes in Cloud Using Ant Colony Optimization,” in *2012 UKSim 14th International Conference on Computer Modelling and Simulation*, 2012, pp. 3–8.

[104] FIWARE: https://www.fiware.org

[105] C. Zhu, X. Li, V. C. M. Leung, X. Hu, and L. T. Yang, “Job Scheduling for Cloud Computing Integrated with Wireless Sensor Network,” in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, 2014, pp. 62–69.

[106] R. Dalvi, “Energy efficient scheduling and allocation of tasks in sensor cloud,” *Masters Theses*, Missouri University of Science and Technology, 2014, 7324.

[107] R. Yao, W. Wang, S. Shin, S. H. Son, and S. I. Jeon, “Competition-based device-to-device transmission scheduling to support wireless cloud multimedia communications,” *Int. J. Distrib. Sens. Networks*, vol. 2014, 2014.

[108] Ling Li, Shancang Li, and Shanshan Zhao, “QoS-Aware Scheduling of Services-Oriented Internet of Things,” *IEEE Trans. Ind. Informatics*, vol. 10, no. 2, pp. 1497–1505, May 2014.

[109] J. Li, Q. Pan, and K. Mao, “Solving complex task scheduling by a hybrid genetic algorithm,” pp. 3440–3443, 2014.

[110] B. Billet, and V. Issarny, “From Task Graphs to Concrete Actions: A New Task Mapping

Algorithm for the Future Internet of Things," In *11th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, 2014, Philadelphia, United States, pp. 470–478, IEEE.

[111] N. Edalat, W. Xiao, N. Roy, S. K. Das, and M. Motani, "Combinatorial Auction-Based Task Allocation in Multi-application Wireless Sensor Networks," in *2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing*, 2011, pp. 174–181.

[112] S. Bhattacharya, A. Saifullah, C. Lu, and G.-C. Roman, "Multi-Application Deployment in Shared Sensor Networks Based on Quality of Monitoring," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, vol. 4, no. 3, pp. 259–268.

[113] C. Wu, Y. Xu, Y. Chen, and C. Lu, "Submodular game for distributed application allocation in shared sensor networks," *Proc. - IEEE INFOCOM*, pp. 127–135, 2012.

[114] Wen Hu, D. O'Rourke, B. Kusy, and T. Wark, "A virtual sensor scheduling framework for heterogeneous wireless sensor networks," in *38th Annual IEEE Conference on Local Computer Networks*, 2013, pp. 655–658.

[115] F. Wang, G. Han, J. Jiang, and H. Qiu, "A Distributed Task Allocation Strategy for Collaborative Applications in Cluster-Based Wireless Sensor Networks," *Int. J. Distrib. Sens. Networks*, vol. 10, no. 6, p. 964595, Jun. 2014.

[116] L. Dai, H. K. Xu, T. Chen, C. Qian, L. J. Xie, and T. Chen, "A Multi-objective Optimization Algorithm of Task Scheduling in WSN," vol. 9, no. 2, pp. 160–171, 2014.

[117] M. Kim and I.-Y. Ko, "An Efficient Resource Allocation Approach Based on a Genetic Algorithm for Composite Services in IoT Environments," in *2015 IEEE International Conference on Web Services*, 2015, pp. 543–550.

[118] S. Kim, "Asymptotic shapley value based resource allocation scheme for IoT services," *Comput. Networks*, vol. 100, pp. 55–63, May 2016.

[119] C. Delgado, J. R. Gallego, M. Canales, J. Ortin, S. Bousnina, and M. Cesana, "An Optimization Framework for Resource Allocation in Virtual Sensor Networks," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–7.

[120] S. Misra, S. Chatterjee, and M. S. Obaidat, "On Theoretical Modeling of Sensor Cloud: A Paradigm Shift From Wireless Sensor Network," *IEEE Syst. J.*, vol. PP, pp. 1–10, 2014.

[121] V. Angelakis, I. Avgouleas, N. Pappas, E. Fitzgerald, and D. Yuan, "Allocation of Heterogeneous Resources of an IoT Device to Flexible Services," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 691–700, Oct. 2016.

[122] M. Vogler, J. Schleicher, C. Inzinger, S. Nastic, S. Sehic, and S. Dustdar, "LEONORE -- Large-Scale Provisioning of Resource-Constrained IoT Deployments," in *2015 IEEE Symposium on Service-Oriented System Engineering*, 2015, pp. 78–87.

[123] M. Aazam and E.-N. Huh, "Resource Management in Media Cloud of Things," in *2014 43rd International Conference on Parallel Processing Workshops*, 2014, vol. 2015–May, pp. 361–367.

[124] S. F. Abedin, M. G. R. Alam, N. H. Tran, and C. S. Hong, "A Fog based system model for cooperative IoT node pairing using matching theory," in *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2015, pp. 309–314.

Appendix A – Energy models

This appendix (A) consists of the detailed energy model used in our experiments to evaluate Zeus. This appendix is referred in Section 4.2, and thus complements the formulation of our MINPP for resource allocation in CoS, providing further detail on it.

The constraint defined per Equation (32), in Section 4.2, concerns the total energy consumption of a VN i during the decision window (comprising all periods p). It considers the energy consumption by all hosts h of VN i (ERA_{ih}). The value of the sum of ERA_{ih} must be smaller than the limit $WEVRA_i$. The ERA_{ih} is defined per Equation (45). It is divided into two portions of energy consumption by host h. The first is used for communication of the output data of VN i (ECM_{ih}). The second is used for processing the request inside VN i (EPR_{ih}). The (ECM_{ih}) is defined by Equation (49) in terms of the fraction of power of host h that is allocated to be dissipated in response to running VN i (P_{ih}) and the sum of all times spent for communication (TTM_i^p) of the VN i in all periods p. The EPR_{ih} is defined per Equation (50) similarly to ECM_{ih}, but using the processing times (PTM_i^p). Thus, we describe P_{ih} through Equations (46), (47) and (48).

As in [12], we use the hypothetical linear power model [96], which describes the power consumption in a host per Equation (46). Equation (46) is divided into two parts: static and dynamic power. Static power P_s is consumed even if the host is idle. The dynamic power $(P_{max} - P_s) \times U$ is proportional to the utilization U of the host, where P_{max} is the maximum power the host can dissipate. We consider that the value of Linear Deviation Ratio (LDR) is zero, thus we have an ideal power model. We also ignore the static power consumption, because it does not vary with the allocation of VNs to requests, reducing the model reduced to Equation (47). Finally, based on Equation (47), Equation (48) describes the fraction of the power allocated by each host h to a VN i in our work.

$$ERA_{ih} = ECM_{ih} + EPR_{ih} \qquad \begin{matrix}\forall i \in SVN \\ \forall h \in SECN\end{matrix} \qquad (45)$$

$$P(U) = (1 + LDR) * [Ps + (Pmax - Ps) \times U] \qquad (46)$$

$$P(U) = (Pmax) \times U \qquad (47)$$

$$P_{ih} = (Pmax_h) \times VECM_{ih} \qquad \forall i \in SVN \qquad (48)$$

$$\forall h \in SECN$$

$$ECM_{ih} = P_{ih} \times \sum_{p=1}^{P} TTM_i^p \qquad \begin{matrix} \forall i \in SVN \\ \forall h \in SECN \end{matrix} \quad (49)$$

$$EPR_{ih} = P_{ih} \times \sum_{p=1}^{P} PTM_i^p \qquad \begin{matrix} \forall i \in SVN \\ \forall h \in SECN \end{matrix} \quad (50)$$

The constraint defined in Equation (33), in Section 4.2, regards the total energy consumption of a PSAN k during the decision window, respective to data updates. Such an energy consumption must be smaller than the limit WEP_k. The sum of y_i^p is the number of data updates of VN i during the decision window. Every data update consumes the same amount of energy of the same PSANs. According to [12], the energy spent by a PSAN k in response for a data update of VN i (EDU_{ik}) is defined per Equation (51).

$$EDU_{ik} = ESE_{ik} + ETX_{ik} + ERX_{ik} + EPR_{ik} + EAC_{ik} \qquad \begin{matrix} \forall i \in SVN \\ \forall \mathrm{k} \in SPSAN \end{matrix} \quad (51)$$

$$ESE_{ik} = b_{ik} \times Vsup_k \times Isens_k \times \xi sens_k \qquad \begin{matrix} \forall i \in SVN \\ \forall \mathrm{k} \in SPSAN \end{matrix} \quad (52)$$

$$ETX_{ik} = b_{ik} \times (Eelec_k + \varepsilon amp_k \times (d_{ik})^2) \qquad \begin{matrix} \forall i \in SVN \\ \forall \mathrm{k} \in SPSAN \end{matrix} \quad (53)$$

$$ERX_{ik} = b_{ik} \times Eelec_k \qquad \begin{matrix} \forall i \in SVN \\ \forall \mathrm{k} \in SPSAN \end{matrix} \quad (54)$$

$$EPR_{ik} = b_{ik} \times [C \times V_{dd}^2 + f]_k \qquad \begin{matrix} \forall i \in SVN \\ \forall \mathrm{k} \in SPSAN \end{matrix} \quad (55)$$

$$EAC_{ik} = n_{ik} \times Eact_k \qquad \begin{matrix} \forall i \in SVN \\ \forall \mathrm{k} \in SPSAN \end{matrix} \quad (56)$$

The energy consumption of PSANs is composed of four major components: sensing, computation, communication and actuation components. Equations (52), (53), (54), (55) and (56) are adapted from our previous work [12] to represent the energy consumption of PSANs. In Equation (52), let $Isens_k$and $Vsup_k$be, respectively, the total current and supply voltage required for sensing activity, while $\xi sens_k$ is the time duration for sensing data from the PSAN and b_{ik} is number of bits collected by the sensing activity. Moreover, we distinguish the energy spent for transmission (ETX_{ik}) and for reception (ERX_{ik}) per Equations (53) and (54). The size

of the transmitted/received data is b_{ik} and d_{ik} represents the distance between transmitter and receiver. The $Eelec_k$ and εamp_k denote energy dissipation of radio and transmission amplifier, respectively. The energy consumption for processing all b_{ik} bits of data (EPR_{ik}) is given by Equation (55), where V_{dd}^2 denotes the thermal voltage of the processor and C and f are processor-dependent parameters. The last part of energy consumption for a device is actuation (EAC_{ik}). This part is hard to estimate because it is highly dependent on the specific actuation task. Equation (56) describes the energy consumption for actuation, where n_{ik} is the number of actuations, each consuming a fixed amount of energy $Eact_k$. For instance, an actuation regarding driving a fan with two motors (possibly in response to temperature variations), then $n_{ik} = 2$.

Finally, we must constraint the energy spent by PSANs also from the perspective of VNs, so that a single VN will not preemptively deplete the batteries of several PSANs subordinated to it. Simply by changing the sum (from index i to index k) in Equation (33), we reach to Equation (34), which regards the total energy consumption of a VN i during the decision window, respective to data updates. This value must be smaller than the limit $WEVDU_i$. For the sake of simplicity, we index this consumption for VNs, although the PSANs are the ones that effectively consume the energy.