JOEL DAVID VALENTE GUERREIRO

VIRTUAL SENSOR NETWORKS: COLLABORATION AND RESOURCE SHARING

UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

2019

JOEL DAVID VALENTE GUERREIRO

VIRTUAL SENSOR NETWORKS: COLLABORATION AND RESOURCE SHARING

PhD Thesis in Computer Science

Work done under the supervision of:
Profª Drª Noélia Correia

2019

Statement of Originality

Virtual Sensor Networks: Collaboration and Resource Sharing

Declaração de autoria de trabalho: Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Candidato:

_____

(Joel David Valente Guerreiro)

Work done at Research Center of Electronics Optoelectronics and Telecommunications (CEOT)

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof[a]Dr[a]Noélia Correia for the continuous support of my PhD and related research, for her patience, motivation, and immense knowledge. Her guidance helped me in all the steps of my research and writing of this thesis. I could not have imagined having a better advisor and mentor for my PhD study.

Secondly, a special thanks to my family, for their support and patience when I was day after day, week after week working on this thesis and had no time left for them, in particular to my two sons, André e Luís, for so many times that I couldn't join and support them in their activities and sports. To my beloved wife, Cristina, for her love, care, patience, support and motivation. Without her precious support it would not be possible to conduct this research.

Finally to my co-workers in the IT Department and University, for the motivation and support.

To all, a very big thanks.

Abstract

This thesis contributes to the advancement of the Sensing as a Service (Se-aaS), based on cloud infrastructures, through the development of models and algorithms that make an efficient use of both sensor and cloud resources while reducing the delay associated with the data flow between cloud and client sides, which results into a better quality of experience for users. The first models and algorithms developed are suitable for the case of mashups being managed at the client side, and then models and algorithms considering mashups managed at the cloud were developed. This requires solving multiple problems: i) clustering of compatible mashup elements; ii) allocation of devices to clusters, meaning that a device will serve multiple applications/mashups; iii) reduction of the amount of data flow between workplaces, and associated delay, which depends on clustering, device allocation and placement of workplaces. The developed strategies can be adopted by cloud service providers wishing to improve the performance of their clouds.

Several steps towards an efficient Se-aaS business model were performed. A mathematical model was development to assess the impact (of resource allocations) on scalability, QoE and elasticity. Regarding the clustering of mashup elements, a first mathematical model was developed for the selection of the best pre-calculated clusters of mashup elements (virtual Things), and then a second model is proposed for the best virtual Things to be built (non pre-calculated clusters). Its evaluation is done through heuristic algorithms having such model as a basis. Such models and algorithms were first developed for the case of mashups managed at the client side, and after they were extended for the case of mashups being managed at the cloud. For the improvement of these last results, a mathematical programming optimization model was developed that allows optimal clustering and resource allocation solutions to be obtained. Although this is a computationally difficult approach, the added value of this process is that the problem is rigorously outlined, and such knowledge is used as a guide in the development of better a heuristic algorithm.

Keywords: Internet of Things, Web of Things, Sensing as-a-Service, Cloud.

Resumo

Esta tese contribui para o avanço tecnológico do modelo de *Sensing as a Service* (Se-aaS), baseado em infraestrutura *cloud*, através do desenvolvimento de modelos e algoritmos que resolvem o problema da alocação eficiente de recursos, melhorando os métodos e técnicas atuais e reduzindo os tempos associados à transferência dos dados entre a *cloud* e os clientes, com o objetivo de melhorar a qualidade da experiência dos seus utilizadores. Os primeiros modelos e algoritmos desenvolvidos são adequados para o caso em que as *mashups* são geridas pela aplicação cliente, e posteriormente foram desenvolvidos modelos e algoritmos para o caso em que as *mashups* são geridas pela *cloud*. Isto implica ter de resolver múltiplos problemas: i) Construção de *clusters* de elementos de mashup compatíveis; ii) Atribuição de dispositivos físicos aos *clusters*, acabando um dispositivo físico por servir múltiplas aplicações/*mashups*; iii) Redução da quantidade de transferência de dados entre os diversos locais da *cloud*, e consequentes atrasos, o que dependente dos *clusters* construídos, dos dispositivos atribuídos aos *clusters* e dos locais da *cloud* escolhidos para realizar o processamento necessário. As diferentes estratégias podem ser adotadas por fornecedores de serviço *cloud* que queiram melhorar o desempenho dos seus serviços.

Foram necessário vários passos com vista ao desenvolvimento de um modelo de negócio de Se-aaS eficiente. Foi desenvolvido um modelo matemático que permitisse analisar o impacto da alocação de recursos na escalabilidade, qualidade de experiência e elasticidade. Foi desenvolvido um modelo matemático inicial para selecionar os melhores *clusters* de elementos de *mashups* (*virtual Things* - coisas virtuais), estando estes pré-calculados. Seguidamente foi proposto um segundo modelo para a criação das melhores *virtual Things* (*clusters* não estão pré-calculados). A avaliação realizada foi efetuada através de heurísticas que têm como base estes modelos. Estes modelos e heurísticas foram desenvolvidos primeiro para o caso em que as *mashups* são geridas pela aplicação cliente, e posteriormente foram extendidos para ocaso em que as *mashups* são geridas pela *cloud*. Com o objetivo de melhorar os resultados obtidos, foi desenvolvido um modelo de programação matemática que determina

os *clusters* (de elementos de *mashup*) e alocação de recursos ótimos. Embora seja uma abordagem computacionalmente exigente, esta tem a vantagem de o problema em particular ficar rigorosamente delineado, o que permitiu desenvolver uma heurística mais eficiente.

Termos chave: Internet das Coisas, Web das Coisas, *Sensing as-a-Service*, *Cloud*.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

5G	:	5th Generation of Wireless Technology
aaS	:	as-a-Service
API	:	Application Programming Interface
CC	:	Cloud Computing
CMC	:	Cheapest Materialization Cost
CoAP	:	Constrained Application Protocol
CoT	:	Cloud of Things
CSP	:	Cloud Service Provider
HCV	:	Highest Cost Variance
HTTP	:	HyperText Transfer Protocol
IaaS	:	Infrastructure as-a-Service
ICT	:	Information and Communication Technology
IoT	:	Internet of Things
LMC	:	Less Materialization Choices
MSaaS	:	Multimedia Sensing as-a-Service
PaaS	:	Platform as a Service
QoE	:	Quality of Experience
QoS	:	Quality of Service
SaaS	:	Software as-a-Service
SAaaS	:	Sensing and Actuation as a Service
Se-aaS	:	Sensing as-a-Service
SN	:	Sensor Networks
SP	:	Service Provider
XaaS	:	Everything as-a-Service
VM	:	Virtual Machine
VSM	:	Virtual Sensor Manager
WSN	:	Wireless Sensor Networks
WoT	:	Web of Things
WWW	:	World Wide Web

Sets

$\mathcal{P}$	:	All properties.
$\mathcal{F}$	:	All functionalities.
$\mathcal{T}^{\mathrm{P}}$	:	Physical Things registered at the cloud.
$\mathcal{T}^{\mathrm{P}}(n)$	:	Physical devices that can be used for the materialization of mashup element n.
$\mathcal{P}_i$	:	Properties of Thing (physical or virtual) i.
$\mathcal{F}_i$	:	Functionalities of Thing (physical or virtual) i.
$\mathcal{N}$	:	All mashup elements.
$\bar{\mathcal{P}}_n$	:	Properties defined for mashup element n.
$\mathcal{T}^{\mathrm{V}}$	:	Virtual Things built by the cloud.
$\mathcal{M}_j$	:	Materialization of virtual Thing j (one or more physical Things).
$\mathcal{M}(f)$	:	Possible materializations for functionality f.
$\mathcal{S}$	:	Distributed computing resources of CSP.
$\mathcal{A}$	:	All applications.
$\mathcal{C}(\mathcal{A}_i)$	:	Independent components of application $\mathcal{A}_i$.
$succ(n)$	:	Successors of mashup element n at the mashup workflow.
τ^i	:	Partition of $\mathcal{T}^{\mathrm{P}}$.
η^i	:	Partition of $\mathcal{N}$.
$\chi(k)$	:	Virtual Things requiring data flow from virtual Thing k.
$\Phi(\mathcal{A}_i)$	:	Virtual Things consumed by application $\mathcal{A}_i$.
$\Psi(k)$	:	Physical Things materializing virtual Thing k.
$\mathcal{G}(\mathcal{N}, \mathcal{L})$	:	Compatibility graph.
$\mathcal{L}$	:	Links in compatibility graph.
$\mathcal{R}$	:	Set of all Thing requests.
$\mathcal{D}(r, i)$	:	Set of Thing requests able to join request $r \in \mathcal{R}$ in materialization at device $i \in \mathcal{T}^{\mathrm{P}}(r)$.

Known Values

f_i	:	Functionality of Thing (physical or virtual) i.
$\bar{f}_n$	:	Functionality defined for mashup element n.
$spo(p)$	:	Semantic description of property p in the form of subject-predicate-object.
$\Phi^{n,n'}$	:	One if $n, n' \in \mathcal{N}$ are incompatible for materialization; zero otherwise.
$\Omega^{n,n'}$	:	One if there is a mashup flow $n \rightarrow n'$, for $n, n' \in \mathcal{N}$; zero otherwise.
$\Delta_{f,i}^{n,p}$	:	Highest gap value, from physical Things enrolled in materialization $\mathcal{M}_i^f \in \mathcal{M}(f)$, for a particular property p of $n \in \mathcal{N}$.
$\Delta^{\max}$	:	Highest possible property gap.
c_r^i	:	Cost of materialization $\mathcal{D}(r, i)$.
δ	:	Probability of being compatible.
Δ_1,...,Δ_5	:	Levels of property requirement to device property gaps.

Functions

$g(\mathcal{T}_j^{\mathrm{P},i})$	:	Considering $\mathcal{T}^{\mathrm{P}}$'s partition τ^i, it gives the virtual Thing associated with element j of τ^i.
$f(\mathcal{N}_j^i)$	:	Considering $\mathcal{N}$'s partition η^i, it gives the virtual Thing associated with element j of η^i.
$spo(p_i)$	:	Subject-predicate-object description of property p_i.
$\Delta(spo(p_n), spo(p_k))$	:	Specifies whether p_n is compatible with p_k, or not.
$(\eta^i, \tau^j)^{\mathrm{SCA}}$	:	Most scalable resource allocation approach for η^i and τ^j.
$(\eta^i, \tau^j)^{\mathrm{ELA}}$	:	Most elastic resource allocation approach for η^i and τ^j.
$(\eta^i, \tau^j)^{\mathrm{QoE}}$	:	Resource allocation approach giving highest QoE for η^i and τ^j.
$h(\eta^i, \tau^j)$	:	Total transfer cost for η^i and τ^j.
$TF^{\mathrm{V2V}}(k, k')$	:	Transfer cost associated with flow between workspaces of virtual Things k and k'.
$TF^{\mathrm{V2A}}(k, \mathcal{A}_i)$	:	Transfer cost associated with flow between workspace of virtual Thing k and application $\mathcal{A}_i$.
$TF^{\mathrm{P2V}}(k', k)$	:	Transfer cost associated with flow between the physical Thing k' and workspace of virtual Thing k.
$prob(k, \mathcal{A}_i)$	:	Probability of virtual Thing k sending its data towards application $\mathcal{A}_i$.
$prob(k, k')$	:	Probability of virtual Thing k having flow towards virtual Thing k'.
$prob(k', k)$	:	Probability of physical Thing k' having flow towards virtual Thing k.

Variables

- $\mathbf{c}_r$: Materialization cost vector.
- $\mathbf{v}_r$: Vector of sets, each set including all requests joining request r in a specific materialization (with cost $\mathbf{c}_r$).
- $\lambda^{\max}$: Highest clique materialization cost (upper bound).
- δ_r^i : One if device $i \in \mathcal{T}^{\mathrm{P}}$ was selected for the materialization of $r \in \mathcal{R}$, zero otherwise.
- α_i^f : One if the i^{th} possible materialization for functionality f, $\mathcal{M}_i^f$, is being used; zero otherwise.
- $\kappa_{f,i}^t$: One if physical Thing $t \in \mathcal{T}^{\mathrm{P}}$ is enrolled in the materialization of virtual Thing $\mathcal{M}_i^f$; zero otherwise.
- $\beta_{f,i}^n$: One if mashup element $n \in \mathcal{N}$ is mapped to virtual Thing $\mathcal{M}_i^f$; zero otherwise.
- $\zeta_{f,i}^p$: Highest gap associated with property p, from all $n \in \mathcal{N}$, at virtual Thing $\mathcal{M}_i^f$.
- $\rho_{f',i'}^{f,i}$: One if there is flow from virtual Thing $\mathcal{M}_i^f$ to virtual Thing $\mathcal{M}_{i'}^{f'}$; zero otherwise.
- Υ : Total gap cost.
- Ψ : Total number of flows.

CHAPTER 1

Introduction

1.1 Motivation and Scope

The Internet of Things (IoT) allows devices/Things to connect and exchange data over the internet. More recently, a move towards the Web of Things (WoT) started to emerge for real world objects to become part of the World Wide Web (WWW) and for resources to be easily discovered, accessed and managed, making Things accessible to web developers around the world [16, 22]. In fact, WoT is envisaged as the key for an efficient resource discovery, access and management of Things. A vital component in such new ecosystem will be the 5th generation wireless technology. 5G offers an optimal telecommunication platform for IoT/WoT to work on, and for this reason the 5G IoT is already called the Internet of Everyone and Everything [52]. 5G technologies meet the requirements of mobile communications and needs for Thing's data to be transmitted, facilitating the emergence of applications integrating multiple physical Things (devices) and virtual resources available at the internet/web.

As more Things become discoverable and accessible in the IoT world, the more it makes sense to rely on cloud infrastructures for storage and processing. Many cloud-based "as-a-Service" (aaS) models have emerged over the last years. The Everything as-a-Service (XaaS) is the term used for this set of service models that aim to concentrate software and hardware resources, offering them as services to a large number of users and, therefore, leveraging utility and consumption of computer resources [14]. The most relevant are:

- **Infrastructure as-a-Service (IaaS)** - provide computer resources (e.g., virtual machines);
- **Platform as-a-Service (PaaS)** - provide computing platforms that may include operating system, database, web server, and others;

- **Software as-a-Service (SaaS)** - where the cloud takes over the infrastructure and platform while scaling automatically.

All these models promote the “pay only for what you use”, where clients can subscribe services provided by Cloud Service Providers (CSP) and pay for the resources/services they use.

The Sensing as-a-Service (Se-aaS) model, also relying on cloud infrastructures, emerges from the previously mentioned WoT and expected increase in the number and type of devices participating in the IoT [28, 41]. Large amounts of data with processing needs will emerge, and new challenges will arise in terms of storage and processing. The Se-aaS, being a cloud-based service model for sensors/data to be shared, allows for a multi-client access to sensor resources, and a multi-supplier deployment of sensors [41]. This way, everyone can benefit from the IoT ecosystem, while benefiting from cloud's storage and processing capabilities.

Besides allowing everyone to benefit from the IoT ecosystem, the Se-aaS model allows highly-available, or resilient, applications to be developed. Basically, resilient applications require planning at both software development and application architecture levels, and Se-aaS platforms can serve as a basis for the last. Besides ensuring robust storage and scalability, such service model allows physical Things to be dynamically allocated to clients/applications because users remain unaware of physical Things involved in the process. That is, the client ends up having no deployment and maintenance costs, while having an on-demand fault tolerant service because clients/applications can always use other available physical Things.

When incorporating Se-aaS platforms in the application architecture, software components end up having bindings to virtual sensors managed at the cloud. Any workflow, wiring together virtual sensors, actuators and services from various web sources, is currently managed at the client side. Such workflows are called mashups and define how the just mentioned elements are wired together. For resources to be used efficiently, multiple mashup elements (from different client applications) should be materialized onto the same physical Thing, if the requested functionality is the same and their property requirements/constraints are not competing. However, managing mashups at the client side will bring significant delays because there are multiple travellings of data to the client side. To avoid this, mashups must be managed at

the cloud and bindings to such mashups must be built.

When software components have bindings to mashups managed at the cloud, approaches for an adequate allocation of resources can be developed. The goal is to use Things/data and cloud resources efficiently, while reducing delays associated with the multiple travellings of data. This is achieved by building optimization models that describe the problem in a compact and non-ambiguous way, ensuring that the optimal solution is obtained for each instance of the problem. Due to the problem's hardness, and consequent difficulty in obtaining solutions for large instances, heuristic algorithms also need to be developed.

1.2 Objectives

The objectives of this thesis are the following:

- To explore clusters/groups of mashup elements, for materialization onto devices. A good clustering and materialization policy is critical for device and cloud resource optimization, but before moving to any specific approach a first step will be performed to unveil the trade-off between scalability, elasticity and Quality of Experience (QoE) when making resource allocation at sensor clouds under the Se-aaS paradigm.

- Clusters/groups of mashup elements can be quite asymmetric in terms of cost (gap between what is requested by mashup elements and physical devices supplied by the cloud). For this reason, a second step is to evaluate the impact of minimizing the highest materialization cost (among all materializations), which can be seen as a fairness approach. Such fairness criteria must be compared against the unfair one (minimizing the sum of all materialization costs, which can lead to asymmetries). This step will assume that groups/clusters of mashup elements are predefined and their materialization is to be decided.

- The next natural step will be to develop specific resource allocation models, for sensor clouds under the Se-aaS paradigm, that build the best clusters of mashup elements and optimize resource usage, while minimizing delays. Resource allocation models will be developed considering both mashups managed at the client side and mashups managed at the cloud.

- Heuristic approaches, for good solutions to be obtained fast, will be explored.

1.3 Contributions

This work has several contributions:

- Evaluation of the impact of resource allocation in scalability, QoE and elasticity, for CSPs to be aware and choose for the best approach according to their specific case.
- A mathematical model is developed to select the best pre-calculated clusters of requests, for them to become virtual Things. The model tries to select devices with properties more close to application requests, leaving devices with higher capabilities idle for future requests.
- A mathematical model is developed for the best clusters to be built (non pre-calculated clusters). Three heuristic approaches, having the mentioned model as a basis, are proposed and evaluated.
- Extension of the mathematical model previously developed, which finds the best clusters considering mashups managed at the client side, for the case of mashups managed in the cloud. The three heuristic approaches are also extended for this purpose.
- A mathematical programming optimization model is developed that is able to obtain the optimal solution in resource allocation.
- Based on the knowledge gained with the development of the mathematical programming optimization model, a new heuristic is proposed. This is able to improve the results obtained by the previous heuristic approaches.

1.4 Thesis Outline

Chapter 1 introduces the Se-aaS paradigm, a cloud based service model that allows a multi-client access to sensor data and multi-supplier deployment of sensors. The objectives of the thesis are also detailed, which mainly focus on the development of resource allocation approaches, based on mathematical models and heuristics, for a good use of resources when responding to client/application mashup requests.

In Chapter 2, the Se-aaS paradigm and the relevance of its use are discussed in more detail. The impact of cloud infrastructure's processing and storage capabilities when responding to data requests, and given a set of registered sensors, is analysed. The related work on IoT, wireless sensor networks, Se-aaS models, architectures and system functionalities is also presented.

Chapter 3 details the model developed to assess the trade-off on scalability, QoE and elasticity, when developing resource allocation models for Se-aaS. This is adequate for scenarios where there are multiple client/application requests and mashups being built.

Resource allocation approaches, defined through mathematical models, are discussed in Chapter 4. Such approaches select (pre-calculated clusters) or build the clusters and virtual Thing materializations based on costs, so that efficient clustering and materialization assignment is performed. These assume mashups being managed by client applications.

A more complete resource allocation approach for Se-aaS clouds embedding mashups into the cloud is detailed in Chapter 5. The models and algorithms from Chapter 4 are extended for this purpose. The idea is that mashups, or parts of it, can be consumed by multiple applications or clients. Besides efficient resource allocation, delays associated with data transfer is minimized. A mathematical programming optimization model and a cluster expansion based heuristic are developed that improve the previously obtained results.

Chapter 6 concludes the thesis report, summarizing the steps and strategies done to improve the resource allocation in the Se-aaS model. This chapter also presents future work.

CHAPTER 2

Sensing as-a-Service Paradigm

2.1 Everything as-a-Service

To understand the as-a-Service models, it is important to see the big picture of Cloud Computing (CC). The CC is a paradigm that aims to provide on-demand computing and storage resources to users, relying on virtualization technologies, while providing Quality of Service (QoS) guaranties. Different aaS models have been presented throughout the years by several authors.

CSPs have already implemented and offer nowadays different services based on CC. These include Infrastructure as-a-Service, providing computing resource services, like processing servers, virtual machines, storage, networks, load balancers, etc.; Platform as-a-Service that delivers complete platforms with operating systems, databases, web services, development environments; and Software as-a-Service, where on-demand software is available at the cloud. A new term was, therefore, introduced that specifies a set of service-oriented architectures and models aiming to concentrate software and hardware resources, to offer services to a large number of users, leveraging utility and consumption of computer resources. Such term is the Everything as-a-Service [14]. All these models promote the "pay only for what you use" while allowing companies to focus on their core competencies, instead of Information and Communication Technologies (ICT) [41] [40].

Sensor based cloud computing emerged more recently and also has CC as a basis. The objective is to make sensor resources/data from Sensor Networks (SN) or Wireless Sensor Networks (WSNs) available to multiple users/clients [33]. A sensor is a device that can measure, detect or sense some phenomenon, like measuring the temperature or light in a room. In [12], the authors describe Sensing and Actuation as-a-Service (SAaaS), where they identify three main actors: Node Owner, CSP and Application Service Providers. The Node

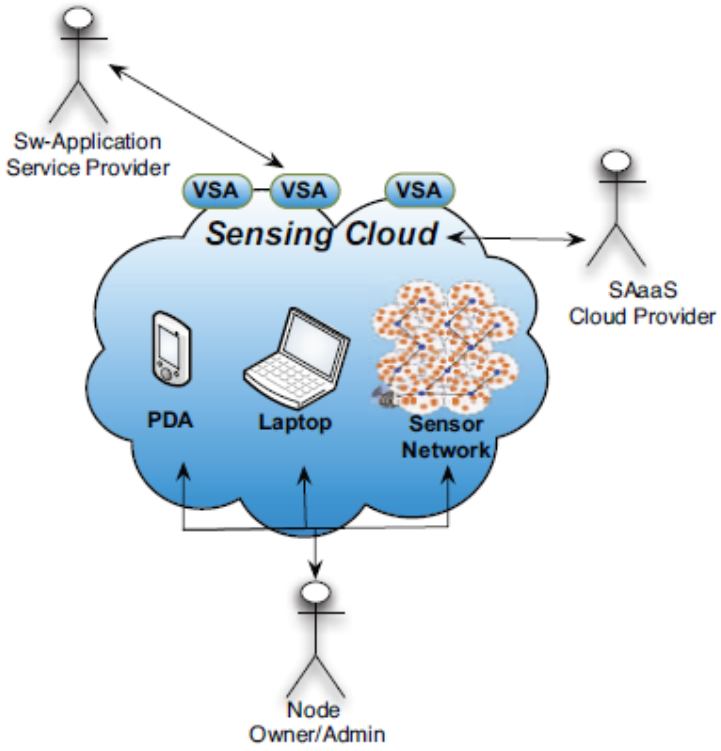


Figure 2.1: SAaaS scenario from [12].

Owner provides the devices into the cloud. The CSP abstracts and virtualizes the devices and provides them as-a-service through an application, used to access data from the devices registered at the cloud. The Application Service Provider is the consumer/client that uses the cloud services (see Figure 2.1).

Se-aaS was first introduced by [45] with the objective of providing sensing services, through a cloud computing environment, using mobile phones. Such work analyses the design of Se-aaS clouds, which should support different applications and be energetically efficient, describes the basic functionalities and tasks and also details the implementation challenges. The authors also introduce several scheduling algorithms to solve energy efficiency problems, and for sensors to become more proficient in loading the sensing data into different smartphone platforms. They also describe the design of an incentive mechanism to attract users to participate with their mobile equipment. This study is the first approach discussing a Se-aaS business model.

The Se-aaS model emerged to assist in the use of resource-constrained Things (devices/sensors), as illustrated in Figure 2.2, and can be used for agriculture, healthcare, environmental purposes, and many other domains. This

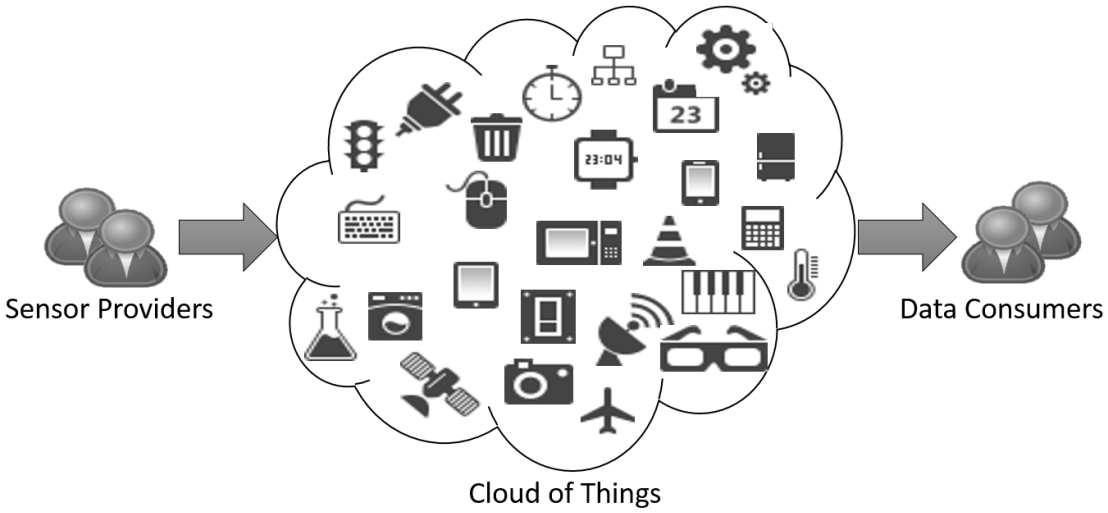


Figure 2.2: Generic Se-aaS environment.

model is able to offer a large number of sensors that users can use without having to own them. They simply subscribe the services from the CSP, and access millions of data from millions of sensors that are registered at the cloud.

Se-aaS is a business model in a CC environment, where sensor owners can register/de-register their devices, providing services to application users that access data in real-time, with fault tolerance, scalability and security. There is a multi-supplier deployment of sensors, and a multi-client access to sensor resources, while the Se-aaS infrastructure provides storage, virtualization and management facilities [38]. A closer look at the Se-aaS model architecture is required to fully understand all details.

2.2 Architecture

The Se-aaS model works on a cloud and has the IoT infrastructure as a service basis, for virtual sensors to become available to users (each virtual sensor is operationalized via one ou more physical devices). This way many users can access data generated from those devices. Similarly to other aaS models, resources should be provisioned and de-provisioned dynamically and on-demand, resulting in a flexible operation. Since sensors (or data) are to be accessed by multiple users/applications in real-time, through service subscription, the costs associated with owing, programming and maintaining sensors and/or sensor networks will scale down [33]. The challenges when planning and designing such systems are the following:

- Underlying complexity should be hidden, so that services and applications can be launched without much overhead;
- Scalability, ensuring a low cost-of-service per consumer while avoiding infrastructure upgrade;
- Dynamic service provisioning for pools of resources to be efficiently used by consumers.

The Se-aaS model consists of three layers: i) Sensor Providers and Physical Sensors; ii) Cloud Infrastructure; iii) Consumers [41]. See Figure 2.3.

- **Sensor Providers and Physical Sensors**: This layer consists of SN, WSN or simply sensors owned by sensor providers (private or public organizations) that publish/register those sensors in the Se-aaS cloud infrastructure and share sensor data. The sensor providers have to manage and control the sensors for them to work perfectly and be able to share data [28].
- **Cloud Infrastructure**: The Cloud Infrastructure layer consists of four types of servers: Portal Server, Data Storage, Monitoring and Management Server (some authors separate these in two different servers), and Provisioning Server.
  - The Portal Server should be able to respond to user's needs when they log in (sensor provider or consumer), allowing sensor providers to register or remove their sensors and consumers to place a request to monitor, manage or terminate virtual sensors and/or virtual sensor groups. Sensor providers should be able to view the usage of their sensors and obtain detailed reports. Consumers should

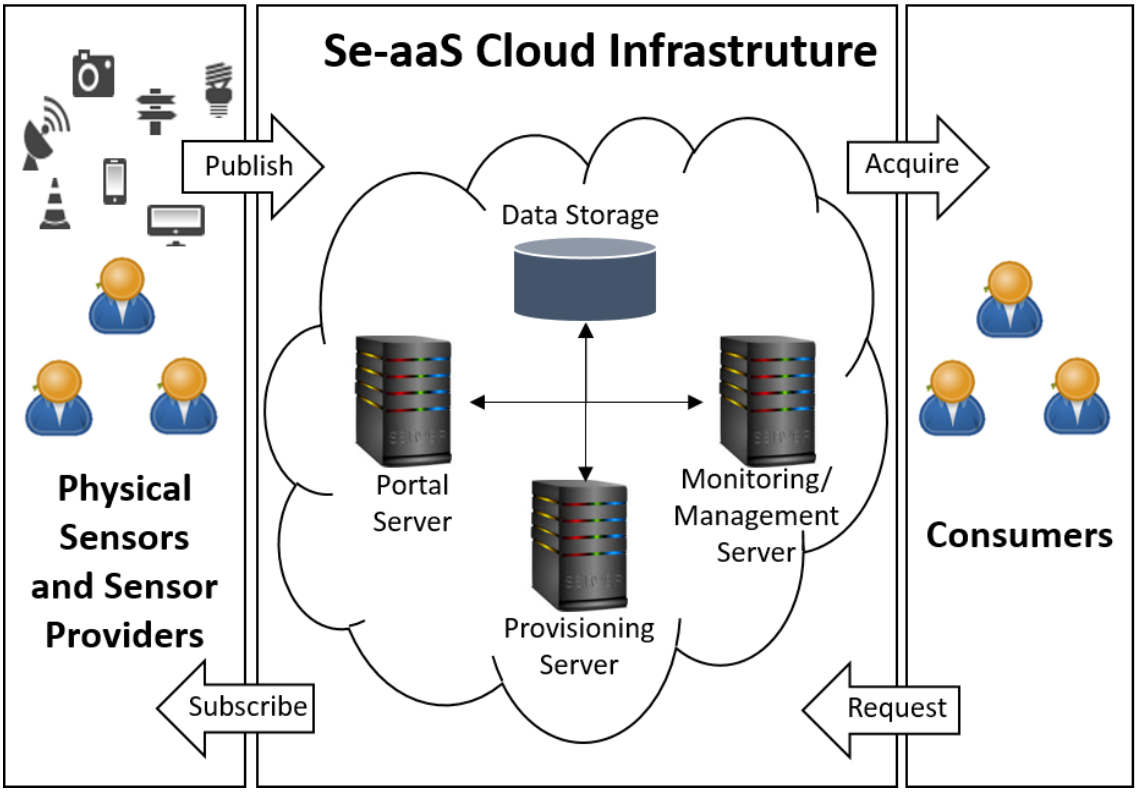


Figure 2.3: Se-aaS architecture.

be able to control and activate/de-activate their subscribed virtual sensors, set how often they want to receive the data and check their status. This server forwards the requests to the Provisioning Server for creation, modification and removal of Virtual Machines (VMs), virtual sensors and virtual sensor groups [28].

- The Data Storage Server ensures the storage of data into the databases, which may relate to user's information, virtual sensors, virtual sensors groups or sensing data [28].
- Monitoring and Management Server is responsible for providing a multi-tenant solution over the cloud to the registered sensor providers and consumers, performing scaling and location-aware load balancing for selected VMs to be closer to the requesting zone. This server is also responsible for retrieving the data and for the health status of the virtual sensors, which is to be stored in the Data Storage Server for virtual sensor information to be available to consumers. This server is extremely important because live data provisioning is based on live physical sensors, and if a physical sensor is offline the Monitoring and Management Server should mark the sensor for this not to be used by the Provisioning Server. This pro-

visioning server should choose another sensor, with the same characteristics and properties, to be binded to the virtual sensor and respond to the consumer requests [28].

  - Provisioning Server is responsible for the creation of virtual sensor groups, and virtual sensors on-demand, according to consumer requests that were registered through the Portal Server. The Provisioning Server can create and reserve VMs when it receives a request from the Portal Server. After the VM is ready, a virtual sensor group can be automatically provisioned and stored in the Storage Server, for the Portal Server to know the virtual sensors that were assigned for that particular request and start receiving the data [28].

- **Consumers**: Consumers will place their requests through the Portal Server, after register themselves, and will receive the real-time analytics they have requested. Consumers may also download archived data and use Application Programming Interfaces (APIs) to connect to their own applications [28].

For a clear understanding of the Se-aaS model, the workflow within this model must be explained. The workflow is initiated with the registration of sensors by the Sensor Providers, through the Portal Server, and information is stored in the Data Storage Server after a status and location checking performed by the Monitoring and Management Server. The consumer registers and inserts its sensing data requests using the Portal Server that will create the virtual sensors, or virtual sensors groups, that can better respond to such requests. This requires using the information stored in the Data Storage Server, which includes the location information determined by the Monitoring and Management server. The Monitoring and Management Server starts then to retrieve the data in real-time and stores it in the Data Storage Server. The results can be visualized by the consumers when using the Portal Server.

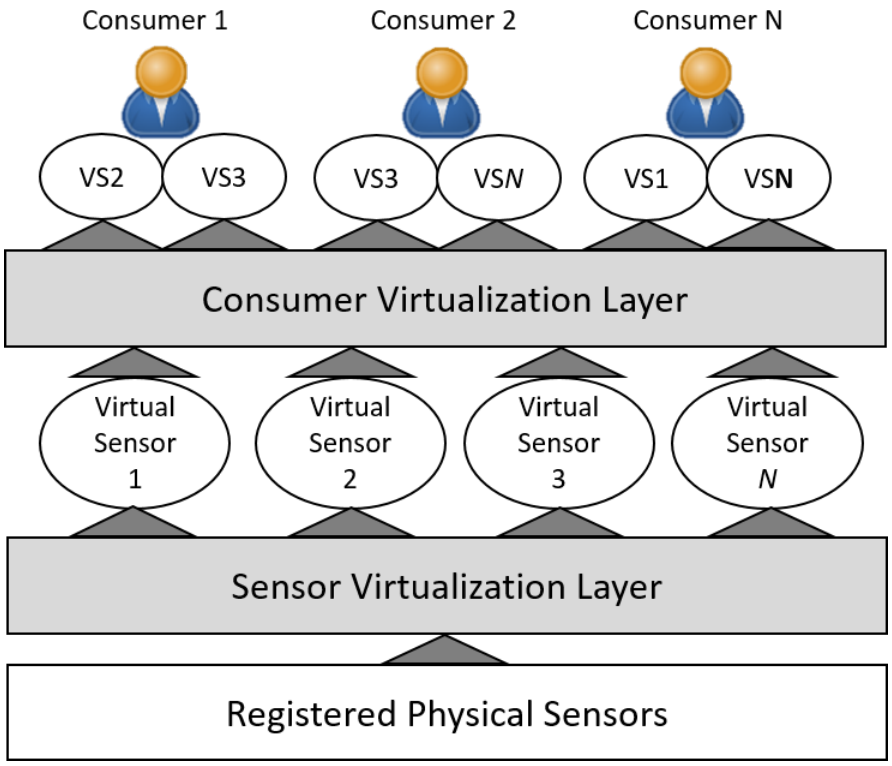


Figure 2.4: Virtualization layers in Se-aaS.

2.3 System Functionalities

Besides registration capabilities for consumers and providers, Se-aaS platforms end up having one or more of the following functionalities [28]:

- **Virtualization**: Sensor virtualization is used to enable the management and customization of devices by clients/applications/consumers, allowing a single device to be linked to one or multiple consumers. Groups of virtual sensors can be made available for specific purposes. Virtualization is illustrated in Figure 2.4.
- **Dynamic Provisioning**: This allows consumers to leverage the vast pool of resources on demand. A virtual workspace (e.g., virtual machine) is usually created for the provisioning of virtual sensors, which can be under the control of one or more consumers. Virtual workspace instances are provisioned on demand, and should be as close as possible to the consumer's zone.
- **Multi-Tenancy**: A high degree of multi-tenancy in architectures allows sharing of sensors and data by consumers, and dedicated instances for each sensor provider. Issues like scaling according to policies, load balancing and security need to be considered.

Figure 2.5: IoT vertical markets and their horizontal integration [5].

2.4 Related Work

2.4.1 Internet of Things

IoT was introduced in 1998 by Kevin Ashton who stated: "*The Internet of Things has the potential to change the world, just as the internet did*" [7]. Then, in 2005, the International Telecommunications Union formally introduced IoT saying: "*from anytime, anyplace connectivity for anyone, we will have connectivity for anything*" [46]. From then on, several projects were launched to provide solutions for implementing IoT on different application domains, like e-Health, smart agriculture, smart water, smart metering, smart environment, security and emergencies, logistics, and so on. Over the last years, cities also emerged as an application domain on which IoT infrastructures are used to implement innovative solutions and services. Thus, data is used for cities to become smart cities.

In Ala Al-Fuqaha et. al., [5], the IoT architecture, main technologies and protocols used, IoT challenges, and the relationship between IoT and other

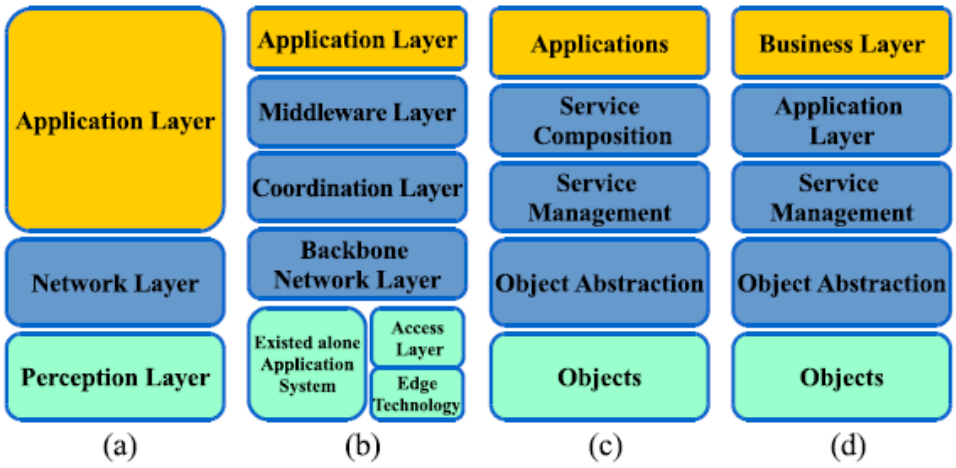


Figure 2.6: IoT architectures: (a) Three-layer architecture; (b) Middleware-based architecture; (c) SOA-based architecture; (d) IoT business model [5].

emerging technologies is discussed. The authors also discuss service use-cases to interconnect different protocols, so that IoT services can be delivered. Their definition of IoT highlights several characteristics: "*The IoT enables physical objects to see, hear, think and perform jobs by having them "talk" together, to share information and to coordinate decisions. The IoT transforms these objects from being traditional to smart by exploiting its underlying technologies such as ubiquitous and pervasive computing, embedded devices, communication technologies, sensor networks, internet protocols and applications. Smart objects along with their supposed tasks constitute domain specific applications (vertical markets) while ubiquitous computing and analytical services form application domain independent services (horizontal markets)*". Figure 2.5 illustrates the interaction between the various application domains, together with the horizontal integration of independent services.

Several IoT architectures have been proposed in the literature, but still no convergence into a single model exists [29]. The authors in [5] summarize IoT architecture proposals discussed in [27, 48, 50] (see Figure 2.6) as follows:

(a) The three-layer architecture of IoT, where the perception, network and application layers are defined;

(b) A middleware-based architecture, where more virtualization is introduced to the model as well as a middleware layer;

(c) SOA-based architecture, which introduces two service layers, for management and composition, to the model as well as an object abstraction layer;

(d) Five layer architecture for an IoT business model.

Miao Wu et al, [48], proposes an additional architecture including the following layers:

- The Perception Layer - This layer is aware of the object properties (temperature, location, etc.) and enables the transmission of such information throughout the networks by converting it into digital signals. There are several techniques used in this layer, such as RFID, 2-D barcodes, GPS, etc. Thus, the main function of the perception layer is to collect and transform information into digital signals for these to be transmitted.

- The Transport Layer - Also called network layer, is responsible for the transmission of the data received by the perception layer throughout the different connected networks. The technologies used in this layer are cellular, Wifi, bluetooth, infrared, etc, and IPv6 is used to address billions of devices. Thus, the main function of the transport layer is to transport information through the various connected networks.

- The Processing Layer - The information received by the transport layer is to be analysed, processed and stored within the processing layer. The main technologies utilized in this layer are: intelligent processing, cloud computing, databases, ubiquitous computing, etc. The large amount of data received from billions of devices makes this layer crucial, and both cloud and ubiquitous computing will have a growing preponderance in the processing and storage of all such data.

- The Application Layer - IoT application development (e.g., intelligent transportation, identity authentication or logistics management) can be achieved using the output data from the processing layer. Thus, the main function of the application layer is to provide applications that can respond to all industries that require such output data, this way promoting the development of the IoT in a larger scale.

- The Business Layer - The main role of this layer is to manage the developed applications, which respond to several industries, and to create a business model that allows profit (using shared data) by several clients.

Rafiullah Khan et all [27], identify the following key challenges in IoT:

- Naming and Identity Management - Each device connected to IoT infrastructures must have a unique identification over the internet, so a naming and identity management system is required to dynamically assign and manage unique identifications in a large scale of objects/devices.
- Interoperability and Standardization - The manufactures provide different technologies and services, creating an ecosystem which is not interoperable and yet not a standard. Standardization is required for the interoperability of all devices/objects.
- Information Privacy - Different technologies are used in devices, like RFID, 2D-barcodes, etc, and it is necessary to ensure privacy and avoid unauthorized access.
- Safety and Security of Objects- Physical damage and intruder's access to devices must be prevented throughout all the geographic areas, avoiding losses of data and objects.
- Data Confidentiality and Encryption - The data transmitted by devices should be encrypted to assure data integrity.
- Network Security - The network should be able to perform data transmission with no data loss due to congestion or security issues, preventing external monitoring or external interference.
- Spectrum - The wireless transmission requires a dedicated spectrum for data to flow normally in the network. Therefore, an efficient cognitive spectrum allocation mechanism is required that allows a large amount of devices to communicate over wireless networks.
- Greening of IoT - Decreasing the energy consumption is the goal and green technologies are required to make devices more efficient.

In [4] the authors present a framework for IoT infrastructures where data is provided via a data cloud for service-based applications. The development of online heuristics for public data delivery in smart cities and a pricing function for data acquisition are presented by the authors. The developed multi-tier framework, able to receive from heterogeneous data sources and to dynamically collect data from peripheral network gateways, receives user requests from access points on top of the architecture and delivers sensor data to consumers. The authors also present a two-tier pricing model, where the first monitors peripheral systems, delays, gateway capacity and system lifetime,

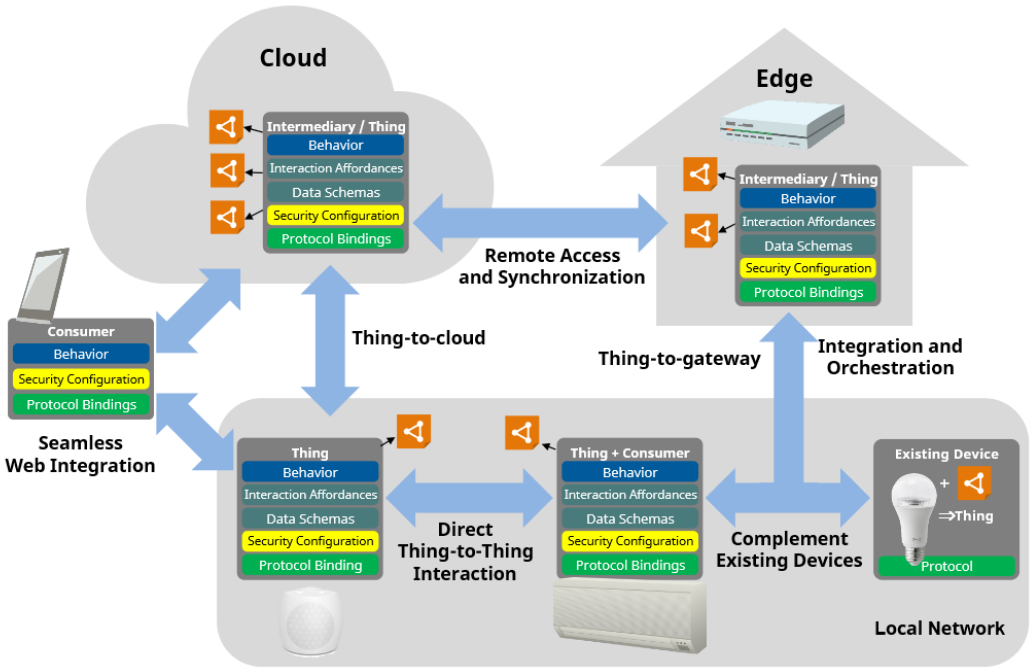


Figure 2.7: WoT abstract architecture [3].

while the second maximizes service quality, trust factor and the monetary value of the data retrieval.

IoT can be defined as a collection of Things (sensors, cellphones, GPS locators, RFID systems or other smart objects) that are identifiable, trackable and connected to the internet [54], providing a large scale access to information. However, many different approaches, applications and ecosystems were developed for IoT, which results in the inability for these to communicate between themselves. Today's reality is that we have a fragmented IoT. In summary, the challenges arising from this fragmented environment are related to [42]:

- Interoperability of different technologies;
- The amount of data generated by IoT devices for processing and deployment;
- How to orchestrate resources from different ecosystems.

2.4.2 Web of Things

The fragmented environment created by IoT, where devices, applications and services are not interoperable, required the emergence of WoT. Sujith S. Mathew

et al., [34], refers to WoT as the application platform for IoT including advances in web services that abstract the underlying technology of Things, enhancing the global interoperability between systems. Soumya K. Data and Cristian Bonnet, [26], presented the interoperability advances in IoT when using WoT. To better understand WoT, the architecture must be presented (See Figure 2.7). The architecture referred by the authors in [3] is composed by three main components:

- The connected device level – where sensors and objects are connected;
- The gateway or edge level – to establish the connection between the connected device level and the cloud level, for data retrieval from physical Things.
- The cloud level – where CC with high levels of processing, storing and services allows the consumers to request and retrieve data.

This architecture allows the connection of web technologies to devices or Things in the internet [23].

WoT allows Things to communicate, collaborate and make decisions autonomously, which can rely on cloud based services [35], resulting in the construction of platforms and applications to collect and process data using web technologies like HTTP, RESTful web services or CoAP. This increases the number of Things connected to the internet and the data retrieved as web resources [9, 43].

Thus, WoT allows the connection of Things to the WWW, enabling their discovery, access and management by users/developers using web protocols [22]. This solves the problem of discovering and connecting Things, but still it is not able to solve the problem of dealing with huge amounts of data, or the problem of how to orchestrate different ecosystems. CC can be the answer to these challenges. The following subsections discuss proposals from the literature that try to address some of these issues.

2.4.3 Wireless Sensor Networks

WSNs are networks capable of connecting distributed autonomous sensors to capture physical and environmental conditions (e.g., temperature, humidity, vibration, motion, pressure, etc.), with the goal to retrieve and manage the

sensors data [53]. Zhu et al. describe in [53] the four main issues when integrating WSNs:

- Location awareness and energy consumption of devices/equipment;
- Authentication and trust calculation;
- Network bandwidth usage to access data;
- QoS improvement.

These issues can be solved with the integration of WSNs into CC infrastructures and by offering sensing services to clients that subscribe them.

Storing and processing the huge amount of data generated by WSNs and IoT devices soon also became a problem. Another solution had to be implemented that could solve these issues, and CC was the answer [53]. The amount of data that can be processed, stored and managed in CC infrastructures is huge due to their capabilities, technology and virtualization techniques. CC infrastructures provide several services for access, sharing and resource utilization. As mentioned before, the most relevant services are now: IaaS, PaaS and SaaS. In the context of WSNs, Deshwal et al. propose in [6] an architecture that introduces IaaS and SaaS to implement an information as a service solution for WSNs. That is, a flexible and reliable infrastructure capable to process, store and secure services targeted to sensor data clients, using virtualization technologies, is proposed.

Regarding virtualization of WSNs, several proposals have appeared in the literature. When virtualizing WSNs, the general idea is that the cloud should abstract different physical device platforms in order to give the impression of a homogeneous network, enhancing user experience when configuring devices. In this context, data storage and/or device assignment to tasks has been recently addressed. In [38] a shift from WSNs to sensor clouds is discussed and a theoretical mathematical model that characterizes the virtualization model is presented. The behaviour of WSN based applications in the sensor cloud platform, using virtualization groups of physical sensors, is studied. See Figure 2.8. The authors presented the previously mentioned architecture from two points of view:

- **User organization's view or logical view** - A user interface is presented as a web page (accessed with a browser) that is running at the site

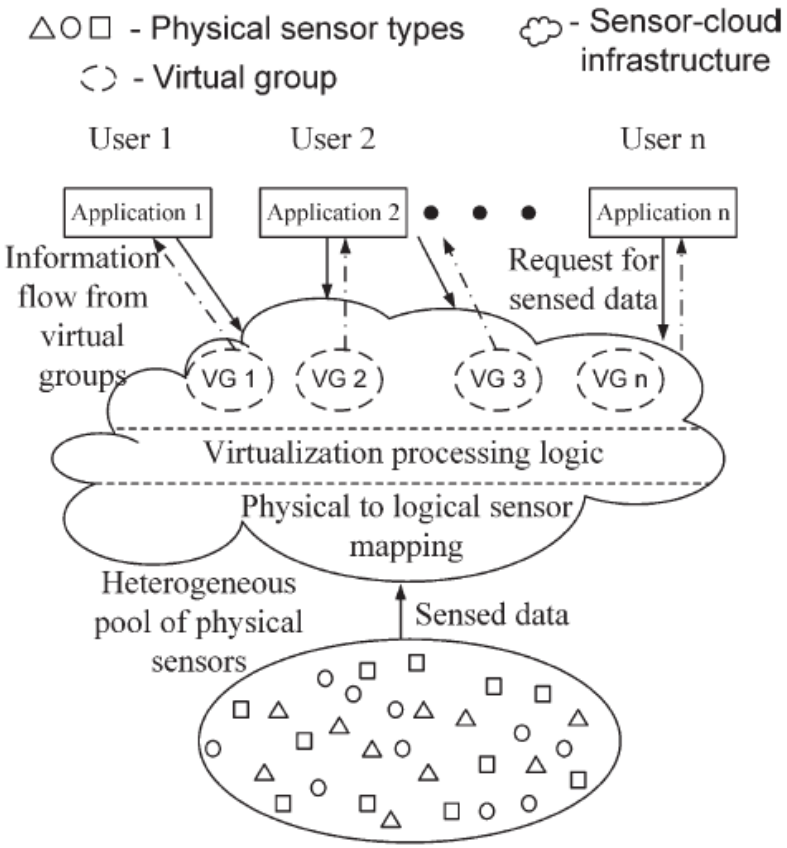


Figure 2.8: Architecture of sensor clouds by S. Misra et al. [38].

of the CSP. An organization (or user) requests virtual sensors through specific templates that collect relevant information, like location or type of sensors. The user is abstracted from all cloud's underlying complexity (processing, sensor node allocation, application aggregation and virtualization) and retrieves the data from the CSP through APIs. Data is then integrated into the application. See Figure 2.9.

- **Algorithmic view or real view** - Figure 2.10 shows the communication flow between users and the entities providing the on-demand allocation of virtual groups of sensors, binded to physical sensors, that can respond to the requests.

Results presented by S. Misra et al. show that the mapping of applications to physical resources, using virtualization of groups of sensors, leads to sensor clouds that outperform traditional WSNs in most of the cases, which justifies the shift to the cloud.

Another proposal regarding the assignment of tasks in WSNs is presented in [53]. Four still ignored research issues are described in the WSN-CC integration:

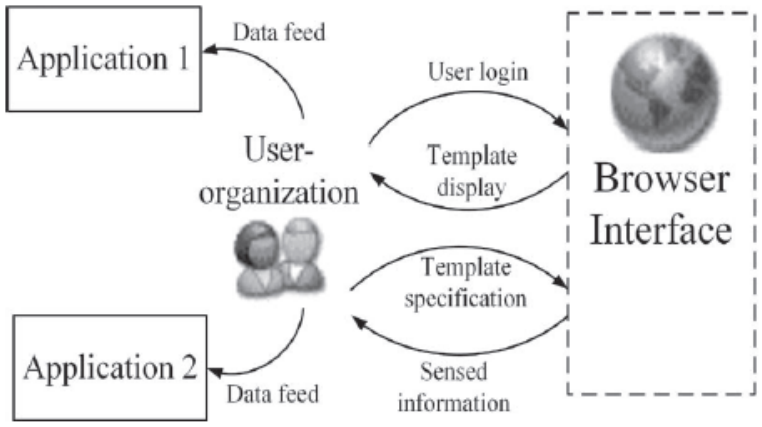


Figure 2.9: View of user organization by S. Misra et al. [38].

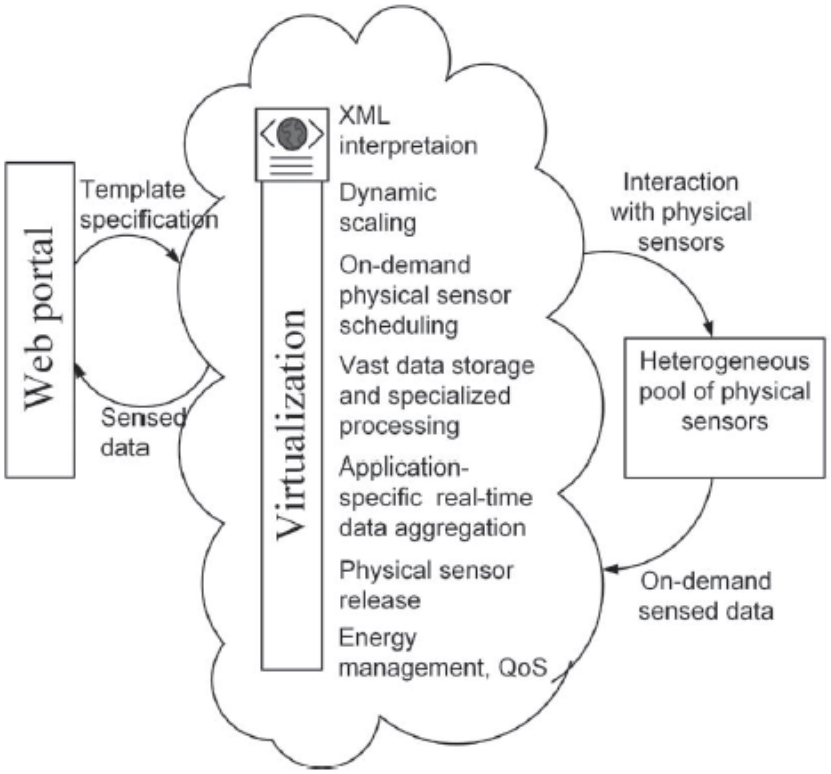


Figure 2.10: Real view of complex processing by S. Misra et al. [38].

- **Sensors usually utilize non-rechargeable batteries with limited energy**. That is, sensor nodes continuously transmit data to the cloud and battery lifetime is short.
- **Authentication of CSP**. False authentications can impersonate users and obtain services from the CSP. Trust and reputation were not yet seriously prevented in CC for WSNs.
- **Applications that require WSNs to reliably offer sensor data**: Useful sensor data should be reliably offered from WSNs to the CC, and

not directly to the users, so that it can be offered to several people.

- **Improving QoS**: Another issue is QoS, where throughput or response time has a vital role for users and should be explored.

The authors respond to these issues with their accomplished work, and for each issue they propose solutions:

- **First issue** - The authors propose two collaborative location based sleep scheduling schemes that consider the location of mobile users, and dynamically change the status of sensor nodes (between awake and asleep) so that they send data only when scheduled, and their battery lifetime increases.
- **Second issue** - The authors propose a novel authenticated trust and reputation calculation considering the authenticity, the CSP and user requirements, and the cost of trust and reputation service.
- **Third issue** - A time and priority based selective data transmission is proposed where just some useful selected data is transmitted to the cloud, considering time and priority features of the data.
- **Fourth issue** - The basic idea is to use trust and enhance WSN-CC integration, where data to the cloud is prioritized over other data.

In [30], data filtering in WSNs, for storage at the cloud, is addressed. The main concept is the use of gateways to link WSNs with the cloud. Gateways collect data from the sensor nodes, perform compressing and then transmit it to the cloud. Neural networks are used to detect anomalies in the gathered data. The authors propose an architecture that reduces sensor energy consumption due to the fact that the majority of data processing is done at the gateways, and also by using scheduled updates to the data at the cloud. The storage at the sensor devices is minimal.

The connection to the cloud solved the issue of storing and processing huge amounts of data in WSNs. Se-aaS emerged from this scenario to provide services to customers paying just for what they use. The next subsections intends to explain Se-aaS evolution throughout the literature.

2.4.4 Sensing as-a-Service

The Se-aaS concept was initially introduced by [4, 45] where authors identify the challenges: different sensing applications should be supported, solutions

should be energy-efficient and an incentive mechanism must exist to attract users to share their sensing activities, using mobile phones (crowd sensing) to fulfil some need or request. The authors in [45] describe the functionalities that a Se-aaS cloud should support:

- **Web Interface**: A web interface to collect the requests from users;
- **Generating Sensing Tasks**: Sensing tasks, containing the request information entered through the web interface, which should follow a standard format;
- **Tracking Mobile Phones**: Information from all mobile phones should be available so that data can be shared. Their location, which sensors are available, energy status, and so on, should be made available. A way to push tasks into mobile phones must exist too, for data to be collected;
- **Recruiting Mobile Users**: A way to recruit mobile phone users, for them to share their mobile sensors, must exist. These should respond to the requested tasks. The recruitment must have an incentive mechanism;
- **Scheduling of Sensing Activities**: An algorithm or policy for the scheduling of sensing activities, at previously recruited mobile phones, must exist for shared data to be collected at the time that is needed;
- **Managing Sensors**: An application in each mobile phone must exist for sensors to be managed, and for scheduled tasks to collect and store data into the cloud;
- **Processing and Storing Data**: A database is required for useful information to be stored, and for data reports to be delivered to the users.

In the context of smart cities, Se-aaS is discussed in [41, 42]. The first addresses technological, economical and social perspectives, exploring the sensing as-a-service concept and how it fits into IoT. The main goal of IoT is to connect devices to the internet, and to allow them to communicate between themselves, and for that to be achieved the devices should have embedded sensing and communication capabilities. This will increase the cost of those devices. Se-aaS is designed to provide incentives to users, so that they are encouraged to participate, and share data, even if it increases the cost of their devices. This is so because users will get a return on their investment. The proposal is to use a Se-aaS model, instead of implementing traditional IoT

smart cities, because a Se-aaS model can be sustainable, scalable and powerful, which allows to efficiently use the limited resources while allowing for a large number of consumers, with a win-win solution for all parties involved. The cities/users have the resources and data they need, and sensor providers have their return according to the envisaged incentives.

The authors in [42] propose a global approach for semantic annotation of sensors at the cloud, allowing different resources to be aggregated. That is, their smart city vision relies on a Cloud of Things (CoT) paradigm. Their main goal is to create a technological agnostic architecture for the integration and deployment of several objects and devices, ignoring their underlying architecture. This must meet all requirements of a smart city. The approach is based on the creation of a platform that envisages the convergence of IoT platforms and ecosystems. That is, CoT creates bridges between different platforms and fragmented ecosystems that otherwise would not be able to communicate. Vertical platforms are horizontally integrated, through a virtualization level that guarantees the semantic interoperability of the different IoT platforms and ecosystems.

In [24, 37], the semantic selection of sensors is addressed. The first proposes a sensing service architecture that is context-aware and applies select and search methods to discover: user preferences, accuracy, power consumption, sensing range, etc. Searches are applied using semantics and parameters/requirements registered by consumers, for sensor selection, and this method proved that lower power consumptions can be achieved, when compared with traditional text based search schemes. The authors of the second work have designed a framework for optimal gateway selection in sensor cloud environments, using semantics, which is applied to remote health monitoring of patients distributed over different geographically locations.

2.4.5 Multimedia Sensing as a Service

Multimedia Se-aaS has been explored in [31, 32, 47, 49]. The authors of the first work have developed a real-time transcoding mechanism with HTTP live streaming in a cloud, using hand-held devices such as mobile phones or tablets. The authors in [49] discuss mobile cloud computing for rich media applications, and their primary goal is to discuss technical challenges like energy consumption, offload computing tasks, limited bandwidth, QoE, privacy and security, and the minimization of costs, using rich media applications in mo-

bile devices with a cloud computing context. They also review some solutions proposed in the literature, like: i) considering energy consumption on the client side, and not on the cloud side (from a green point of view, however, the energy consumption should be implemented on both sides and a solution is still to be investigated); ii) adaptive QoE provisioning, which is an open problem for scheduled sessions in wireless networks and in cloud environments; iii) security and privacy, since video applications developed in the mobile cloud were still not secure and privacy was not yet stabilized and fully implemented. The authors suggest that those solutions should be investigated.

In [32], a network and device QoS-aware approach for cloud based mobile streaming is presented. Their main objective is to provide multimedia data, suitable for terminal units using interactive mobile streaming services provided by a cloud, while adjusting the interactive transmission frequency and the dynamic multimedia transcoding, so that bandwidth and power waste is avoided. A prototype of this architecture is implemented, together with an efficient self-adaptive service for multimedia streaming depending on different bandwidth environments.

Wang et al., in [47], explores the resource saving potential of cloud-edge IoT and Fogs for Multimedia Sensing as a Service (MSaaS). In such article, the MSaaS concept is introduced for the first time. Their contribution is the development of a MSaaS resource allocation framework for cloud edges and fogs. They analyse frequency, data dependencies, temporal domains and interaction to optimize the resource allocation at cloud-edge IoT and Fogs.

2.4.6 WSN and IoT Service Models

Service-centric models in [12, 51] focus on the services provided by a WSN acting as a service provider, and not on the WSN virtualization for an homogeneous network to be built. The authors in [51] explain how IoT and Big Data connects with Se-aaS, due to the volume, variety and velocity of sensor data that needs to be processed, extracted (useful information) and stored with high performance. The authors explain that Big Data is not just the size of data generated, but also the variety, different types of data, and how frequently data is generated (occasionally, frequently or in real-time). Techniques are essential for managing and storing sensor data, so that it can be presented as-a-Service.

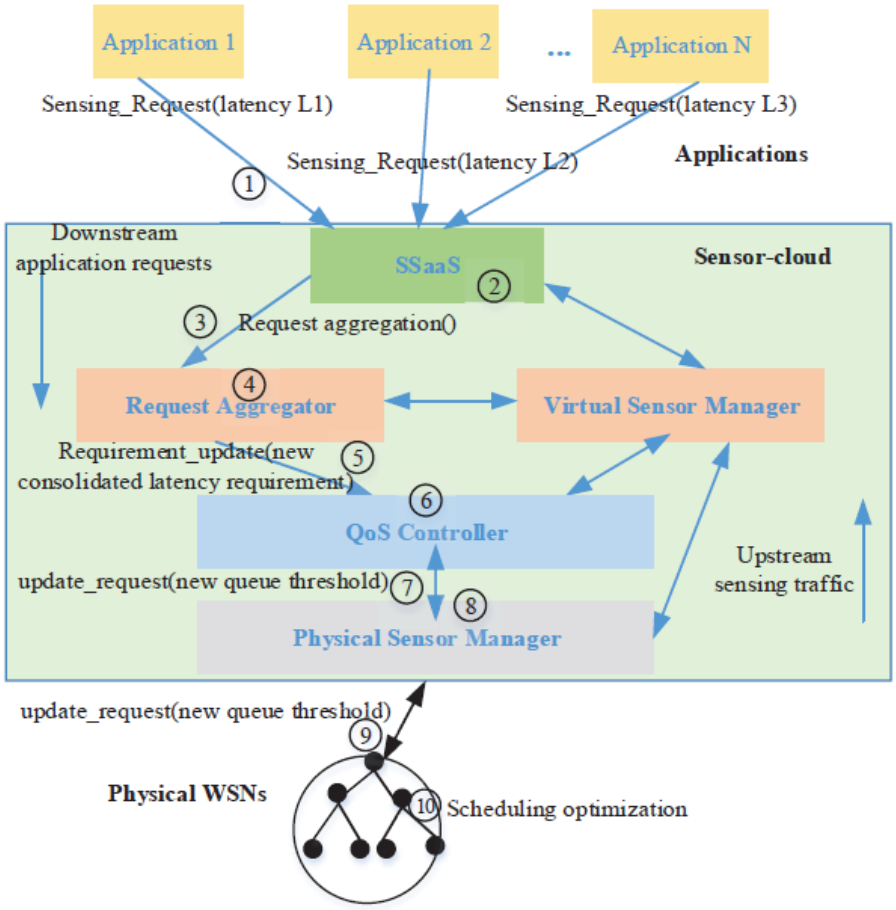


Figure 2.11: Interactive model by T. Dinh et al. [11].

The authors in [12] focus on the implementation of an as-a-Service infrastructure, through CC, where wireless sensor nodes in a WSN can be managed and aggregated dynamically, so that new resources become available as-a-Service.

Regarding IoT service models, the general idea is to virtualize sensing services provided by devices. A virtual sensor ends up being responsible for passing user's specifications to device(s) and for processing the sensed data before delivering it to users. In [11] an interactive model is presented where the aggregation of multiple application requests, with the objective of minimizing workloads, control latency and save energy, is addressed (see Figure 2.11). The sensor cloud consists in virtualizing sensors in a cloud environment that is responsible for providing services to the applications, considering the user specifications to retrieve data (e.g., sensor types, locations or QoS requirements). The model consists in top-down (application requests) and bottom-up (sensing traffic) streams. The sensor cloud in this model is considered a middleware between the physical sensors and the applications,

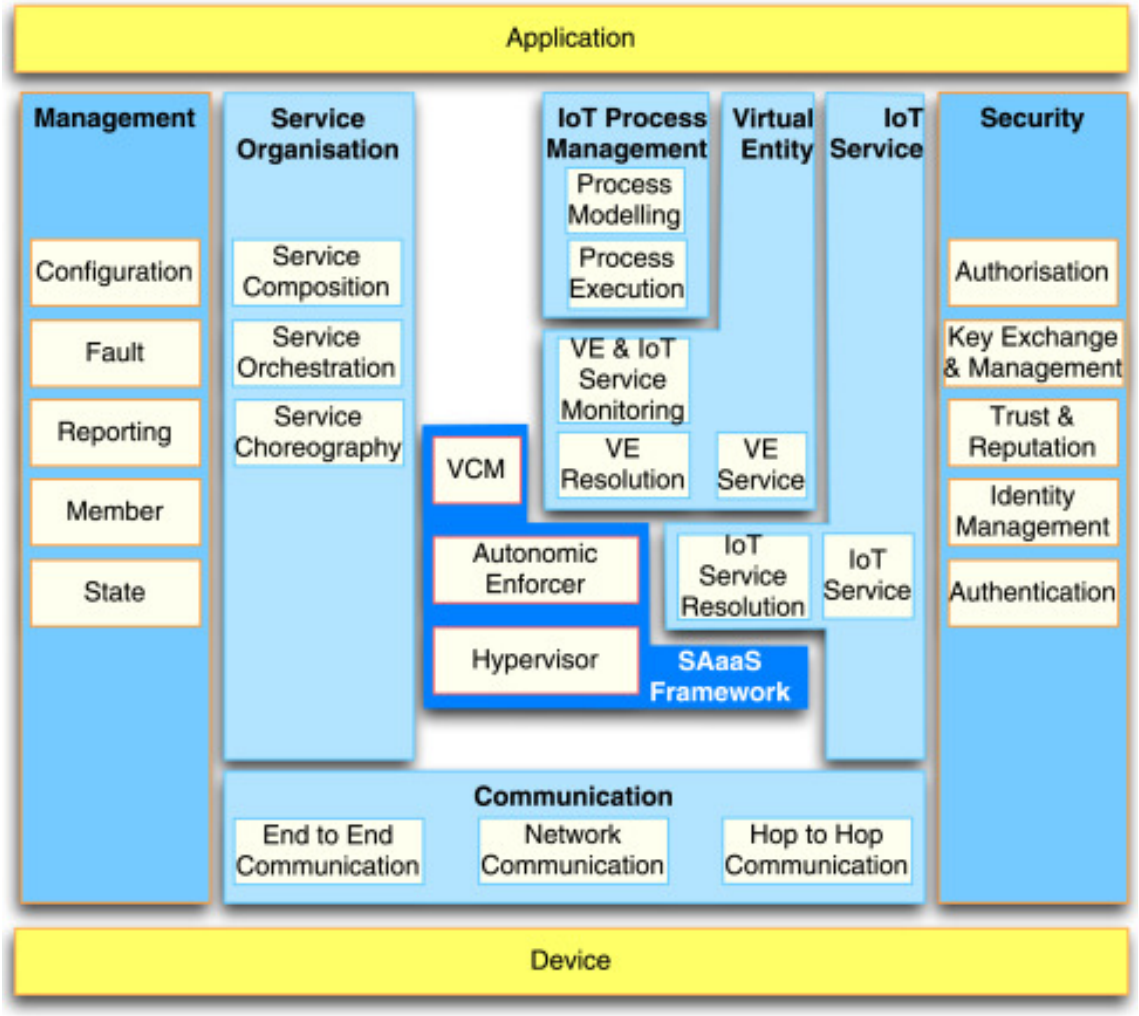


Figure 2.12: Sensor cloud implemented by S. Distefano et al. [13].

virtualizing physical into virtual sensors. Results show that this model effectively controls latency of sensor flows and applications at the same time, providing high scalability.

The authors in [13], similarly to [11], design and develop sensing abstraction and virtualization functionalities in sensing CC. A sensing cloud is implemented that aggregates sensing resources and personal mobile devices, which were virtualized, and then provides as a service to end users (see Figure 2.12). Tests were done with Android platform through a typical IoT application to demonstrate the feasibility of the architecture. Results show that this approach is feasible [13]. In summary, [11, 13] present solutions for the physical resources to be abstracted, virtualized and presented as a service to the end users. This way, the access and interaction with physical Things becomes uniform and in compliance with IoT/WoT goals.

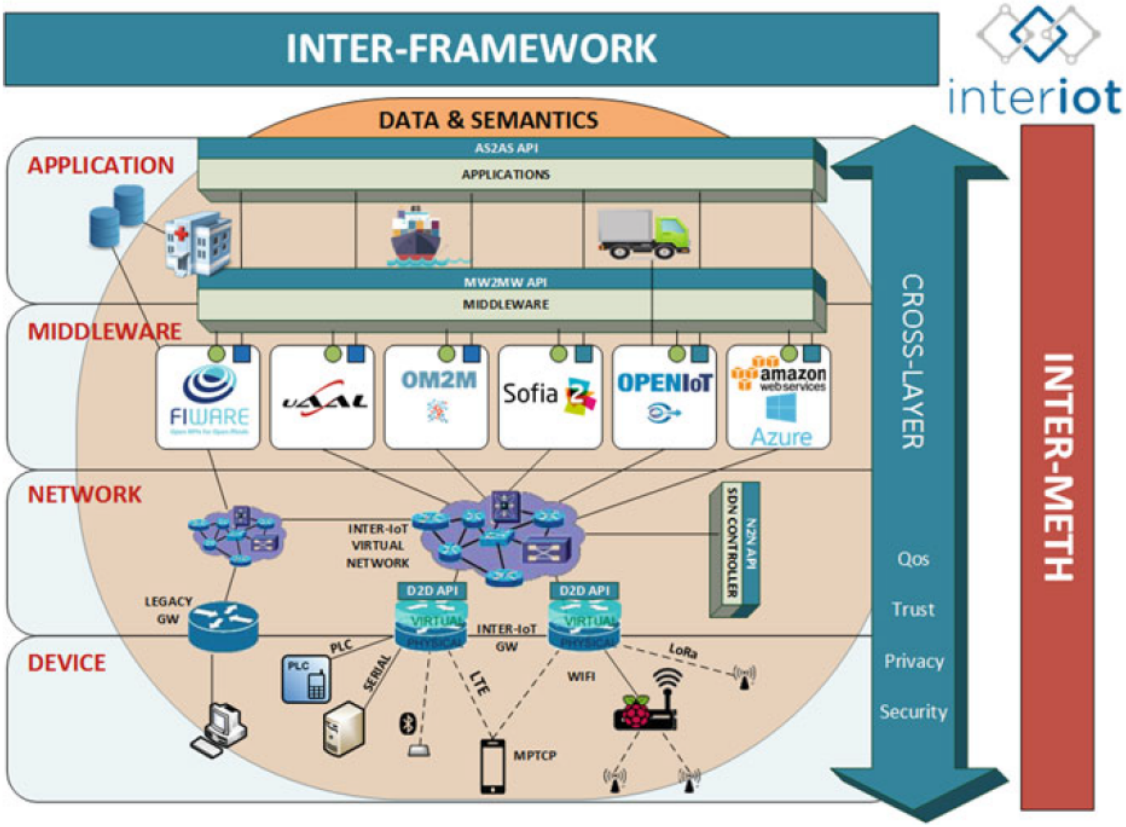


Figure 2.13: INTER-IoT abstract architecture by G. Fortino et al. [15].

Specific platforms providing efficient sharing mechanisms for data (among multiple applications) were proposed in [15, 25, 36]. [25] propose a new concept for distributed sensor networks sharing , using a virtual federated sensor network. In the absence of IoT standards, the INTER-IoT systemic approach was presented in [15], which aimed to provide ways to overcome interoperability issues between heterogeneous IoT systems in devices, networks, middleware, application services and data/semantics (see Figure 2.13).

The architecture describes three main solutions that allows interoperability in a voluntary way:

- **INTER-Layer** - A set of tools and methods, to provide interoperability between layers, using virtual gateways/devices to enable communication between devices, networks and middleware's. A broker for data and semantics interoperability is also presented in this solution.
- **INTER-FW** - A framework for managing and programming IoT platforms. An API to access and manage the ecosystem of IoT applications and services is presented in this solution, which also is aimed to provide security and privacy features, as the creation of a community of users

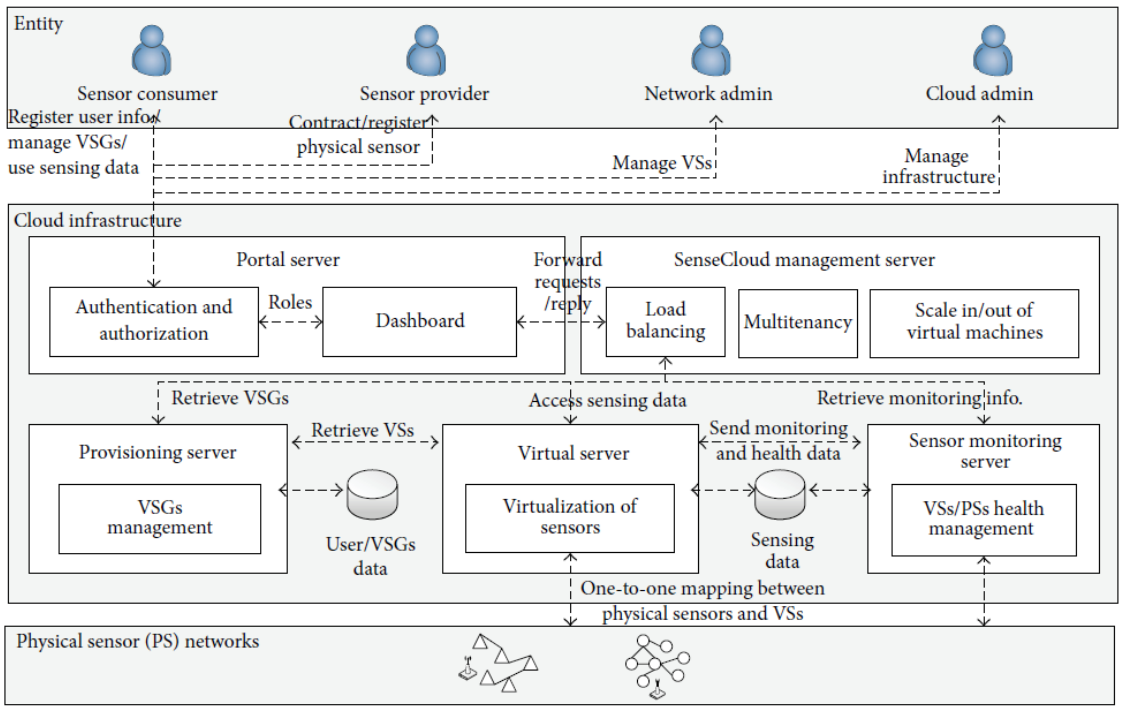


Figure 2.14: Sensor cloud system architecture by M. Kim et al. [36].

and developers too.

- **INTER-Meth** - Computer Aided Software Engineering (CASE) tool with the objective of systematically drive the integration/interconnection of heterogeneous non-interoperable IoT platforms.

The architecture presented by the authors enables the creation of an ecosystem of different devices, networks and middleware to interoperate between themselves and to provide data sharing.

A cloud based solution was developed by [36] with the goal to provide a standardized access to different sensor networks, abstracting the underlying complexity. *SenseCloud* implements a sensor virtualization mechanism for the connection of several sensor networks, a multi-tenancy mechanism granting access to virtualized sensor networks, and a dynamic provisioning mechanism that allows on-demand user requests with a pay per user basis (see Figure 2.14).

The SenseCloud architecture, [36], consists of three main components (Entity, Cloud Infrastructure and Sensor Network) and four entities (Sensor Consumer, Sensor Provider, Network Admin and Cloud Admin). The roles of each entity are the following:

- **Sensor Consumer** - Registers on the cloud and subscribes the interest-

ing sensors or creates sensor groups for data retrieval.

- **Sensor Provider** - Registers himself and the sensors in the cloud. It is possible to manage, control and check sensor status. Sensor Providers can also access sensor usage.
- **Network Admin** - Monitors sensor health and manages the virtual sensors and provider accounts.
- **Cloud Admin** - Manages and monitors VMs, cloud Infrastructure, consumer accounts and services available.

The cloud infrastructure includes a set of servers, all having different roles:

- **Portal Server** - It is the entity's main access, each entity having different operations available. For example, Sensor Consumers may be able to subscribe or create sensor groups or to retrieve data from those sensors, or may be able to register or remove sensors and manage the sensor status. Another example is the cloud admin that is able to create, modify or remove VMs, virtual sensor and virtual sensor groups.
- **Provisioning Server** - The main purpose of this server is to create virtual sensor groups, depending on the requests of Sensor Consumers inserted through the Portal Server. It is also responsible for the workflow of the system, triggering VMs and virtual sensor groups that respond to the specified needs, and updating records in data storage with the virtual sensor groups created, so that Sensor Consumers can manage, activate/deactivate the subscribed virtual sensors or virtual sensor groups, set the frequency of data retrieval and check sensor status.
- **Sensor Monitoring Server** - Receives the informational status data from virtual sensors and stores it, to become available to Sensor Consumers. It also receives the health status of physical sensors because on-demand provisioning is done on live physical sensors.
- **Management Server** - Provides the location of the closest VM to the request sender zone which has the shortest pending list and a multi-tenant solution for the Sensor Consumers and Providers. It also provides scaling depending on the network and system performance.
- **Virtual Server** - Creates virtual sensors when requested by the Provisioning Server, on the VM. Virtual sensors are controlled by the Portal

Server. Also provides health status about the sensors to the Monitoring Server and stores it in Data Storage.

- **Data Storage** - Consists in databases for users, virtual sensors groups and sensing data.

The SenseCloud infrastructure enables connections of different networks, resolves connectivity concerns and efficiently provides Se-aaS (through cloud) between Sensor Providers and Consumers.

CHAPTER 3

Resource Allocation Trade-offs in Sensing as-a-Service

3.1 Introduction

In this chapter, an initial mathematical model is developed to evaluate resource allocation tradeoffs in sensor clouds. One of the first sensor cloud models proposed in the literature is the one in [38]. However, their focus is on WSNs and on how these can move to sensor clouds, not being adequate for other IoT Se-aaS business models. More specifically, in [38] sensors are allocated to a single application and mashups are not addressed. Therefore, it can be seen as a WSN virtualization.

Here, a model suitable for emerging IoT related Se-aaS business models, including the one in [38] is proposed. This model considers sensors/data sharing by multiple applications and mashups, allowing one to access the impact of resource allocation approaches (both cloud and physical Things), and better understanding of trade-offs, so that mechanisms can be orchestrated to face future requests.

Contributions

- A mathematical model that is able to deal with multiple requests, from multiple applications, while considering mashups.
- Evaluation of the impact of resource allocation in scalability, QoE and elasticity, for CSPs to be aware and choose for the best approach according to their specific case.

These contributions were published in:

- J. Guerreiro, L. Rodrigues and N. Correia, “Modelling of Sensor Clouds Under the Sensing as a Service Paradigm”, Proceedings of *Broadband*

Communications, Networks and Systems (BroadNets), September 2018, [20].

3.2 Definitions and Assumptions

Definition 1 (Physical Thing)**.** *A sensor detecting events/changes, or an actuator receiving commands for the control of a mechanism/system. The model of a physical Thing i includes all properties necessary to describe it, denoted by $\mathcal{P}_i$, and all its functionalities, denoted by $\mathcal{F}_i$. That is, $\mathcal{P}_i = \{p : p \in \mathcal{P}\}$, where $\mathcal{P}$ is the overall set of properties (e.g., sensing range, communication facility, location), and $\mathcal{F}_i = \{f : f \in \mathcal{F}\}$, where $\mathcal{F}$ is the overall set of functionalities (e.g., image sensor), considering all devices registered at the cloud.*

It is assumed that properties and functionalities, at $\mathcal{P}$ and $\mathcal{F}$ respectively, result from a semantic description of physical Things registered at the cloud. That is, specific vocabularies are used when naming properties and functionalities (see [10], for example). Each property $p_i \in \mathcal{P}_i$ has a "subject-predicate-object" description[1] denoted by $spo(p_i)$ (e.g., temperature hasValue $30°$C). The set of all physical Things is denoted by $\mathcal{T}^{\mathrm{P}}$, and sensor owners voluntarily register/deregister physical Things to/from the cloud.

Definition 2 (Mashup)**.** *Workflow built by wiring together Things and services from various web sources, on which an application is based.*

That is, applications (at the user side) should be able to access Things at the cloud and, if necessary, blend them with other services and data sources on the web, as shown in Figure 3.1. However, for resources to be used efficiently, applications should not pick physical Things directly. Instead, a functionality requirement and minimum/maximum property requirements should be specified for each element n included in a mashup, denoted by $\bar{f}_n$ and $\bar{\mathcal{P}}_n$, allowing later an optimized allocation of physical Things to mashup elements. Each $p_n \in \bar{\mathcal{P}}_n$ can have a "subject-predicate-object" description of the condition/requirement that is being defined (e.g., cameraResolution greaterThan 12.1MP; frequencySampling equalTo 10s), denoted by $spo(p_n)$. The overall population of mashup elements (from all applications) at the cloud will be denoted by $\mathcal{N}$.

For devices/data to be consumed by multiple applications, virtual Things will be created at the cloud. Then, each mashup element is binded to a single

[1] A Resource Description Framework (RDF) triple.

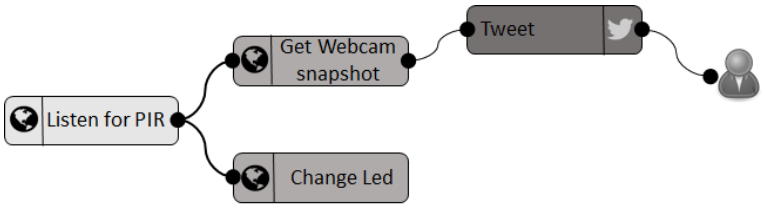


Figure 3.1: Thing mashup.

virtual Thing, while a virtual Thing can be binded to multiple mashup elements (with same functionality and compatible property requirements). Basically, virtual Things represent multiple mashup elements, from multiple applications, and these are the ones to be materialized onto physical Things. Such an approach allows data generated by a virtual Thing to be consumed by multiple applications, while reducing data collection/storage and increasing data utility. Mashup elements are, however, application dependent.

Definition 3 (Virtual Thing)**.** *Thing at the cloud to which mashup elements are binded to. A virtual Thing j is materialized through one or more concerted physical Things, denoted by $\mathcal{M}_j$, $\mathcal{M}_j \subset \mathcal{T}^P$, able to provide the requirements associated with the virtual Thing (requirements from all mashup elements binded to it). Therefore, $f_j \triangleq \cup_{i \in \mathcal{M}_j} \mathcal{F}_i$ and $\mathcal{P}_j = \cup_{i \in \mathcal{M}_j} \mathcal{P}_i$.*

That is, a virtual Thing can have one or multiple physical Things in the background working together. The set of all virtual Things is denoted by $\mathcal{T}^{\mathrm{V}}$.

With virtualization users remain unaware of the physical devices used, allowing these to be dynamically allocated to virtual Things. The client ends up having no deployment and maintenance costs, while having an on-demand fault tolerant service because virtual Things can always use other available physical Things.

A CSP, denoted by $\mathcal{S}$, includes a set of distributed computing resources, each set serving a certain region or having a certain role. Therefore, $\mathcal{S} = \{\mathcal{S}_1, ...\mathcal{S}_{|\mathcal{S}|}\}$. The set of applications (outside the cloud), requesting for sensors with certain properties, is denoted by $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_{|\mathcal{A}|}\}$. An application $\mathcal{A}_i$ can have one or more independent components, denoted by $\mathcal{C}(\mathcal{A}_i) = \{\mathcal{C}_1^i, ...\mathcal{C}_{|\mathcal{C}(\mathcal{A}_i)|}^i\}$, and each component $\mathcal{C}_j^i$ is binded to a mashup (at the cloud) of δ_j^i steps, $\mathcal{C}_j^i \triangleq \{1, ..., \delta_j^i\}$. The following is also assumed:

3.2 Definitions and Assumptions

- Web templates are used to draw the mashup associated with each component, where minimum/maximum property and functionality requirements are specified for each mashup element. Elements can be connected, and $succ(n)$ denotes the successors of element n at the mashup workflow (elements to which n sends data to).
- Final mashups data is sent to the corresponding application components through bindings.
- Virtual Things are created, and binded to mashup elements, by the cloud.

3.3 Mathematical Model

One or more physical Things materialize one virtual Thing. Assuming $\tau^i = \{\mathcal{T}_1^{\mathrm{P},i}, \mathcal{T}_2^{\mathrm{P},i}, ...\}$ to be a partition of $\mathcal{T}^{\mathrm{P}}$, function $g : \tau^i \rightarrow \mathcal{T}^{\mathrm{V}}$ is defined for virtual Thing materialization:

$$g(\mathcal{T}_j^{\mathrm{P},i}) = \{\exists! k \in \mathcal{T}^{\mathrm{V}} : f_k \triangleq \cup_{l \in \mathcal{T}_j^{\mathrm{P},i}} \mathcal{F}_l\}. \tag{3.1}$$

This states that a virtual Thing $k \in \mathcal{T}^{\mathrm{V}}$ is mapped to $\mathcal{T}_j^{\mathrm{P},i}$ if they are functionally similar. Assuming now $\eta^i = \{\mathcal{N}_1^i, \mathcal{N}_2^i, ...\}$ to be a partition of $\mathcal{N}$ (all elements in $\mathcal{N}_j^i$ with the same functionality requirement), function $f : \eta \rightarrow \mathcal{T}^{\mathrm{V}}$ is defined to bind $\mathcal{N}_j^i$ to a virtual Thing:

$$f(\mathcal{N}_j^i) = \{\exists! k \in \mathcal{T}^{\mathrm{V}} : \bar{f}_n = f_k \wedge \bar{\mathcal{P}}_n \subseteq \mathcal{P}_k \wedge \Delta(spo(p_n), spo(p_k)) = \mathrm{true}, \\ , \forall n \in \mathcal{N}_j^i, \forall p_n \in \bar{\mathcal{P}}_n, \forall p_k \in \mathcal{P}_k\}, \tag{3.2}$$

where Δ specifies whether p_n is compatible with p_k, or not. This states that a virtual Thing $k \in \mathcal{T}^{\mathrm{V}}$ mapped to $\mathcal{N}_j^i$ must: i) provide the functionality being requested by elements in $\mathcal{N}_j^i$; ii) fulfill the property requirements of all elements in $\mathcal{N}_j^i$.

Different resource allocation approaches (partitions and allocations done by g and f) can be adopted by sensor clouds, each with an impact on scalability, elasticity and QoE. Let us assume that η^{U} is the universe set of all feasible partitions of mashup elements, $\eta^{\mathrm{U}} = \{\eta^1, \eta^2, ..., \eta^{|\eta^{\mathrm{U}}|}\}$ and $\eta^i = \{\mathcal{N}_1^i, \mathcal{N}_2^i, ..., \mathcal{N}_{|\mathcal{T}^{\mathrm{V}}|}^i\}$, $\forall i \in \{1, ..., |\eta^{\mathrm{U}}|\}$. Also, τ^{U} is the universe set of all feasible partitions of physical Things, $\tau^{\mathrm{U}} = \{\tau^1, \tau^2, ..., \tau^{|\tau^{\mathrm{U}}|}\}$ and $\tau^i = \{\mathcal{T}_1^{\mathrm{P},i}, \mathcal{T}_2^{\mathrm{P},i}, ..., \mathcal{T}_{|\mathcal{T}^{\mathrm{V}}|}^{\mathrm{P},i}\}$, $\forall i \in \{1, ..., |\tau^{\mathrm{U}}|\}$. Thus, each element in τ^{U} is a feasible materialization of virtual Things. For such universe sets, the most scalable resource allocation approach (system can accommodate more load/clients in the future) would select the following solution:

$$(\eta^i, \tau^j)^{\mathrm{SCA}} = argmin_{\eta^i \in \eta^{\mathrm{U}}}\{|\eta^i|\}. \tag{3.3}$$

That is, since each element of partition η^i will be associated with a virtual Thing, fewer virtual Things not only means less virtual workspaces but also more productive virtual Things, as data flowing from them serves more mashups/ applications.

Elasticity is the ability to adapt resources to loads. That is, resources

should become available when the load increases, but when the load decreases then unneeded resources should be released. Thus, the most elastic resource allocation approach would select the following solution:

$$(\eta^i, \tau^j)^{\mathrm{ELA}} = argmin_{\eta^i \in \eta^{\mathrm{U}}}\{max_{\mathcal{S}_l \in \mathcal{S}}\{\sum_{k \in \mathcal{T}^{\mathrm{V}}} \xi(k, \mathcal{S}_l)\}\}, \tag{3.4}$$

where $\xi(k, \mathcal{S}_l)$ is the amount of computational resources allocated to virtual Thing k at $\mathcal{S}_l$. Therefore, virtual Things are evenly distributed by CSPs, which means that any virtual Thing can still work even if its associated load increases.

Regarding the resource allocation approach with a better impact on the QoE perceived by users, this would be the one selecting the following solution:

$$(\eta^i, \tau^j)^{\mathrm{QoE}} = argmin_{\eta^i \in \eta^{\mathrm{U}}, \tau^j \in \tau^{\mathrm{U}}}\{h(\eta^i, \tau^j)\}, \tag{3.5}$$

where $h : \eta^{\mathrm{U}} \times \tau^{\mathrm{U}} \rightarrow \Re^+$ is a cost function defined as:

$$h(\eta^i, \tau^j) = \sum_{k \in \mathcal{T}^{\mathrm{V}}} \sum_{k' \in \chi(k)} TF^{\mathrm{V2V}}(k, k') + \sum_{\mathcal{A}_i \in \mathcal{A}} \sum_{k \in \Phi(\mathcal{A}_i)} TF^{\mathrm{V2A}}(k, \mathcal{A}_i) + \\ + \sum_{k \in \mathcal{T}^{\mathrm{V}}} \sum_{k' \in \Psi(k)} TF^{\mathrm{P2V}}(k', k). \tag{3.6}$$

The $TF^{\mathrm{V2V}}(k, k')$ is a transfer cost associated with the data flow between the workspaces of virtual Things k and k' at the cloud (Virtual-to-Virtual cost), because mashup elements (mapped to virtual Things k and k') can be connected. The $\chi(k)$ must provide all virtual Things k' requiring data flow from virtual Thing k. That is,

$$\chi(k) = \{k'' \in \mathcal{T}^{\mathrm{V}} : k = f(\mathcal{N}_l^i), k'' = f(\mathcal{N}_m^i) \wedge \ n' \in succ(n), n \in \mathcal{N}_l^i, \\ , n' \in \mathcal{N}_m^i, \mathcal{N}_l^i, \mathcal{N}_m^i \in \eta^i\}. \tag{3.7}$$

The $TF^{\mathrm{V2A}}(k, \mathcal{A}_i)$ is a transfer cost associated with the data flow between the workspace of virtual Thing k and the user application $\mathcal{A}_i$. The $\Phi(\mathcal{A}_i)$ provides all virtual Things consumed by application $\mathcal{A}_i$,

$$\Phi(\mathcal{A}_i) = \{k' \in \mathcal{T}^{\mathrm{V}} : k' = f(\mathcal{N}_l^i) \wedge succ(n) = \emptyset \wedge n \in \mathcal{C}_j^i, \mathcal{N}_l^i \in \eta^i, n \in \mathcal{N}_l^i, \\ , \mathcal{C}_j^i \in \mathcal{C}(\mathcal{A}_i)\}. \tag{3.8}$$

Finally, the $TF^{\mathrm{P2V}}(k', k)$ is a transfer cost associated with the data flow between

the physical Thing k' (or its corresponding proxy/gateway) and the workspace of virtual Thing k, which depends on the materialization of k. Therefore, $\Psi(k)$ will be

$$\Psi(k) = \{k'' \in \mathcal{T}^{\mathrm{P}} : k = g(\mathcal{T}_j^{\mathrm{P},i}) \wedge k'' \in \mathcal{M}_k, \mathcal{T}_j^{\mathrm{P},i} \in \tau^j\}, \tag{3.9}$$

which is basically the set of physical Things materializing virtual Thing k. Regarding the transfer cost itself, this may include the number of hops, processing required at the destination, etc, or any combination of these.

3.4 Analysis of Results

3.4.1 Scenario Setup

A set of random graphs, using the algorithm in [39], were used to apply the model described. These graphs, each with 10 nodes, represent the location of CSP's resources, $\mathcal{S}_1, ...\mathcal{S}_{|\mathcal{S}|}$. There are $|\mathcal{A}| = \kappa_1 \times |\mathcal{S}|$ applications and $|\mathcal{T}^{\mathrm{P}}| = \kappa_2 \times |\mathcal{S}|$ physical Things registered at the sensor cloud, where κ_1 and κ_2 are integers. Each $\mathcal{S}_i \in \mathcal{S}$ connects, on average, $\frac{|\mathcal{A}|}{|\mathcal{S}|}$ applications and $\frac{|\mathcal{T}^{\mathrm{P}}|}{|\mathcal{S}|}$ physical Things to the cloud.

The virtual Things to be built depend on physical Things, application requirements and aggregation level when allocating mashup elements to virtual Things. Therefore, tests were done for different amounts of virtual Things, $\frac{|\mathcal{A}|\times\kappa_3\times\kappa_4}{10} \leq |\mathcal{T}^{\mathrm{V}}| \leq \frac{|\mathcal{A}|\times\kappa_3\times\kappa_4}{2}$, where κ_3 is the average number of components per application and κ_4 is the average number of elements at mashups. For transfer costs in Eq. (3.6), the following is assumed:

- TF^{V2A}: Since there will be κ_3 bindings of data flow from the cloud to an application, a virtual Thing k will send its data towards application $\mathcal{A}_i$ with probability $prob(k, \mathcal{A}_i) = \frac{\kappa_3}{|\mathcal{T}^{\mathrm{V}}|}$.
- TF^{V2V}: Since each mashup has $\kappa_4 - 1$ flow links[2], a virtual Thing k has a data flow towards k' with probability $prob(k, k') = \frac{(\kappa_4-1)\times\kappa_3\times|\mathcal{A}|}{|\mathcal{T}^{\mathrm{V}}|\times(|\mathcal{T}^{\mathrm{V}}|-1)} \times \alpha$, where α is the virtual Thing sharing factor or ratio $\frac{\kappa_4\times\kappa_3\times|\mathcal{A}|}{|\mathcal{T}^{\mathrm{V}}|}$.
- TF^{P2V}: A physical Thing k' has a data flow towards virtual Thing k with probability $prob(k', k) = \frac{\kappa_5}{|\mathcal{T}^{\mathrm{P}}|}$, where κ_5 is the average number of physical Things in a virtual Thing materialization.
- The number of hops is assumed to be the transfer cost in TF^{V2A}, TF^{V2V} and TF^{P2V}.

Table 3.1 shows the parameter values assumed.

3.4.2 Discussion

Figure 3.2 shows[3] the impact of $|\eta^i|$ (or number of virtual Things), which is a consequence of the aggregation level used by resource allocation approaches. Less virtual Things means that solutions are more scalable.

[2] A flow tree is assumed.

[3] Average of results obtained for all generated graphs.

Table 3.1: Simulation setup to evaluate the impact in scalability, QoE and elasticity.

Parameter	Value
Number of nodes at CSP graph ($\|\mathcal{S}\|$)	10
Number of applications ($\|\mathcal{A}\|$)	30
Number of physical Things ($\|\mathcal{T}^{\mathrm{P}}\|$)	30
Avg number of components per app (κ_3)	3
Avg number of elements at each mashup (κ_4)	3
Virtual Thing materialization factor (κ_5)	1
Lowest number of virtual Things	$\frac{\|\mathcal{A}\|\times\kappa_3\times\kappa_4}{10}$
Highest number of virtual Things	$\frac{\|\mathcal{A}\|\times\kappa_3\times\kappa_4}{2}$

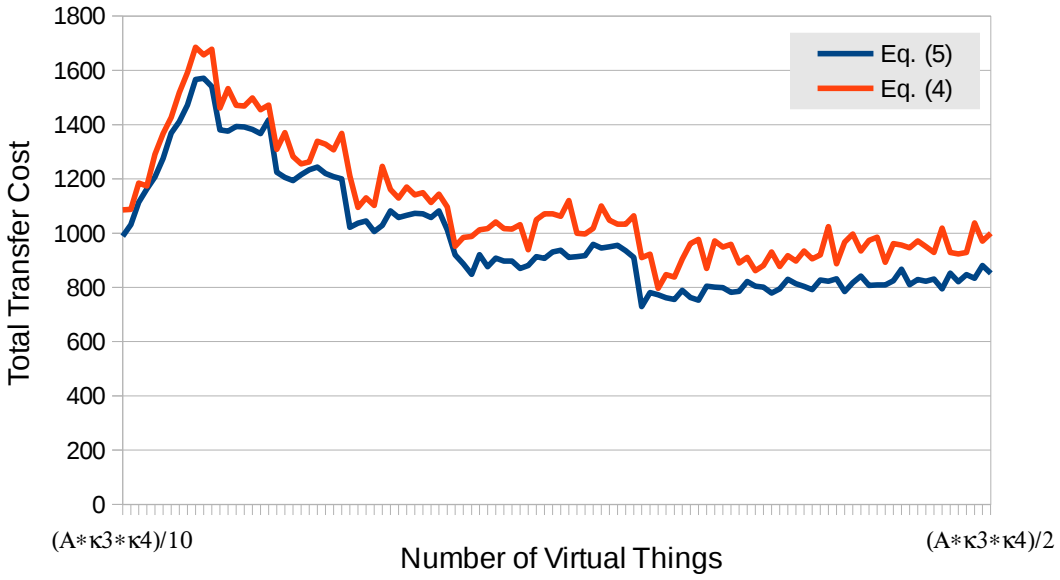


Figure 3.2: Impact of $|\eta^i|$ (number of virtual Things).

A relevant observation regarding the impact of making more or less scalable choices (virtual Things serving more or less applications), is that in general the QoE and elasticity improve as sensor clouds choose for less scalable solutions. In this case, virtual Things are serving less applications and, therefore, less data transfers between virtual Things occurs and data takes less hops to flow towards applications. Also, for each virtual Thing there will be less load. However, this does not happen for a small number of virtual Things. In this case, increasing the number of virtual Things leads to a higher transfer cost, with a negative impact on QoE, and worse elasticity. This happens because virtual Things are already highly dependent, and flow from physical Things towards the cloud takes over the previously mentioned benefit of using more virtual Things. Thus, scalability can have a positive or negative impact on QoE and elasticity depending on the scenario (mashups, etc), which will

determine possible allocations of mashup elements to virtual Things, and the best resource allocation approach to use.

3.4.3 Conclusions

A model for sensor clouds is presented allowing the impact of resource allocation to be assessed, and trade-off between scalability, QoE and elasticity to be unveiled. Results show that the best resource allocation approach is highly dependent on mashups, which will influence possible allocations of mashup elements to virtual Things. This awareness allows sensor cloud providers to choose the best approach according to their case.

The next chapter discusses strategies for mashup element clustering (building virtual Things) and materialization.

CHAPTER 4

Resource Allocation in Sensing as-a-Service: Clients Managing Mashups

4.1 Introduction

The problem of choosing for the best clustering and materialization assignment is addressed here in this chapter, having the associated cost as a basis of decision. Two different variants of this problem are addressed: i) pre-calculated potential clusters; ii) non pre-calculated clusters. Mathematical models are developed for both these problems.

Regarding non pre-calculated clusters, different strategies to guide the search of the best clustering were considered: i) Cheapest Materialization Cost (CMC); ii) Less Materialization Choices (LMC); Highest Cost Variance (HCV). A performance analysis of these different strategies is done. This chapter addresses no mashup element dependencies, meaning that mashup elements (of mashups managed at the client side) can be seen as a pool of individual requests from the different users. That is, it is assumed that mashups are being managed at the client side.

Contributions

- A mathematical model is developed to select the best pre-calculated clusters of requests, for them to become virtual Things. The model tries to select devices with properties more close to application requests, leaving devices with higher capabilities idle for future requests.
- A mathematical model is developed for the best clusters to be built (non pre-calculated clusters). Three heuristic approaches, having the mentioned model as a basis, are proposed and evaluated.

These contributions were published in:

- J. Guerreiro, L. Rodrigues and N. Correia, “Fair Resource Assignment at Sensor Clouds under the Sensing as a Service Paradigm”, Proceedings of *Technological Innovation for Resilient Systems* (DOCEIS), January 2018 (Vol 521, Springer), [19].
- J. Guerreiro, L. Rodrigues and N. Correia, “On the Allocation of Resources in Sensor Clouds under the Sensing as a Service Paradigm”, submitted to *HCI International Conference on Human-Computer Interaction 2020*, [18].

4.2 Pre-calculated Potential Clusters

In the following sections the overall set of physical devices is denoted by $\mathcal{T}^{\mathrm{P}}$, and each $i \in \mathcal{T}^{\mathrm{P}}$ is assumed to have a single functionality and one or more properties (sensor owners voluntarily register/deregister physical Things to/from the cloud). Client applications build their mashups, each with one or more nodes/elements, at the client side. Each mashup element is a request to the cloud and the overall set of requests is denoted by $\mathcal{R}$, and each $r \in \mathcal{R}$ specifies a functionality requirement and one or more property constraints. As devices are registered, the cloud is able to:

- maintain a graph $\mathcal{G}(\mathcal{R}, \mathcal{L})$, where $\mathcal{R}$ includes all requests and $\mathcal{L}$ denotes a set of links. A link between r_i and $r_j \in \mathcal{R}$ exists if: i) requests have similar functionality requirements; ii) property requirements are not incompatible.
- update, for each $r \in \mathcal{R}$, a cost vector $\mathbf{c}_r = \left\{c_r^1, c_r^2, ..., c_r^{|\mathcal{T}^{\mathrm{P}}|}\right\}$ where c_r^i is the cost of using device i for the materialization of the maximum clique, in graph $\mathcal{G}(\mathcal{R}, \mathcal{L})$, that includes r. Such clique is largest set of compatible requests. This way, high clustering of requests is ensured.

Assuming that gaps, between a property requirement and device property supply, are normalized using $\{\Delta_1, ..., \Delta_5\}$, where Δ_1 is the lowest cost and Δ_5 is the highest (moderate and extreme levels), a cost c_n^i will be the sum of all property requirement to device property gaps. If more than one request at the clique has a requirement for a certain property, then the lowest one is chosen (e.g., assuming requests for 12.1MP and 24.2MP camera resolutions, and a physical Thing providing 48.4MP, then the 24.2 to 48.4MP gap is the one to be considered; the other request is considered to be fulfilled).

Several materialization possibilities, with different costs, are being provided as input information. The problem is which materializations to choose. Here the minimization of the highest cluster-Thing assignment cost is analysed. The goal is to see if (contrarily to an unfair approach where the overall cost is minimized) this leads to assignments with lower property gaps, allowing physical Things with features that are above what is requested to be left for future requests and/or backup. This makes applications more resilient.

4.2.1 Problem Formalization

The following information is assumed to be known:

$\mathcal{R}$	Set of all requests.
$\mathcal{T}^{\mathrm{P}}$	Set of physical devices.
$\mathcal{T}^{\mathrm{P}}(r)$	Set of physical devices that can be used for the materialization of request $r \in \mathcal{R}$.
$\mathcal{D}(r,i)$	Set of requests (clique) able to join $r \in \mathcal{R}$ in materialization at device $i \in \mathcal{T}^{\mathrm{P}}(r)$.
c_r^i	Cost of materialization $\mathcal{D}(r,i)$.

Since multiple components contribute to a single materialization cost (i.e., there are multiple properties and, therefore, gaps between physical properties and requirements), and to fairly compare the clusters, c_r^i is divided by the number of summed up property gaps. The variables will be:

$\lambda^{\max}$	Highest clique materialization cost from selected materializations.
δ_r^i	One if device $i \in \mathcal{T}^{\mathrm{P}}$ was selected for the materialization of $r \in \mathcal{R}$, zero otherwise.

- Objective function:

$$OF^{\text{Fair}} : Minimize\ \lambda^{\max} + \frac{\sum_{r\in\mathcal{R}}\sum_{i\in\mathcal{T}^{\mathrm{P}}} c_r^i \times \delta_r^i}{|\mathcal{R}| \times |\mathcal{T}^{\mathrm{P}}| \times \Delta^{\max}} \tag{4.1}$$

where $\Delta^{\max}$ is the highest c_r^i. The first component is used for fairness, because $\lambda^{\max}$ will be an upper bound, while the second component is used to minimize the overall cost when the upper bound can not become lower.

Subject to:

- Upper bound limitation:

$$\sum_{i\in\mathcal{T}^{\mathrm{P}}(r)} c_r^i \times \delta_r^i \leq \lambda^{\max}, \forall r \in \mathcal{R} \tag{4.2}$$

- Single materialization for a request:

$$\sum_{i\in\mathcal{T}^{\mathrm{P}}(r)} \delta_r^i = 1, \forall r \in \mathcal{R} \tag{4.3}$$

- Materialization of cliques:

$$\delta^i_{r'} \geq \delta^i_r, \forall r \in \mathcal{R}, \forall i \in \mathcal{T}^{\mathrm{P}}(r), \forall r' \in \mathcal{D}(r,i) \tag{4.4}$$

- Allocation of device to a single clique materialization:

$$\sum_{r' \in \mathcal{R} \setminus \mathcal{D}(r,i)} \delta^i_{r'} \leq (1 - \delta^i_r) \times |\mathcal{R}|, \forall r \in \mathcal{R}, \forall i \in \mathcal{T}^{\mathrm{P}}(r) \tag{4.5}$$

- Non-negative variables

$$\delta^i_r \in \{0,1\}; \lambda^{\max} \geq 0. \tag{4.6}$$

4.2.2 Analysis of Results

Scenario Setup

To evaluate the allocation of resources, a pool of 10 functionalities were generated for the cloud, each with its own pool of 10 properties. Based on these, a population of physical Things and requests were generated as follows:

- Each physical Thing has a randomly selected functionality, each functionality including 100% of the properties from the corresponding pool.
- The request functionality requirement is also randomly selected from the pool of functionalities, together with 50% of its properties. Each pair $r, r' \in \mathcal{R}$ sharing the same functionality requirement is assumed to be compatible with probability of $\delta = 0.5$ or $\delta = 0.75$ (creation of cliques).
- The gap between a property requirement and device property supply is randomly selected from $\{\Delta_1 = 1, \Delta_2 = 2, ..., \Delta_5 = 5\}$ (moderate and extreme levels).

The population of requests is 50 and tests were done for 80, 90, 100 and 110 physical Things. The CPLEX [1] optimizer was used to solve this problem.

Fair vs Unfair Approach

To evaluate the impact of the fair approach, a second objective function was also implemented. This second objective function has no fairness into consideration (no minimization of the upper bound is performed) and chooses to reduce the overall cost only. It is defined as follows:

[1] IBM ILOG CPLEX Optimizer

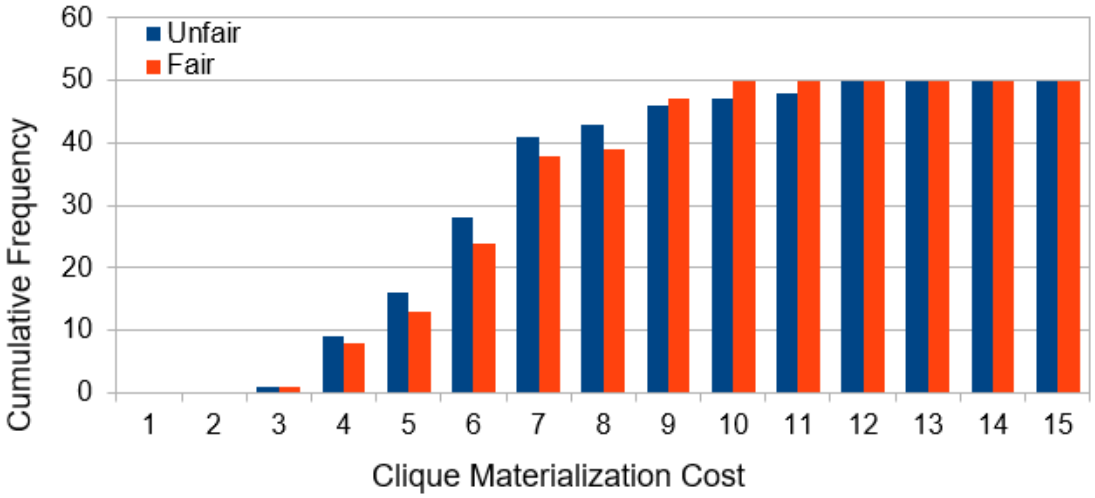


Figure 4.1: Selecting pre-calculated potential clusters: Cumulative frequency considering 90 available physical Things (mashups managed by the client).

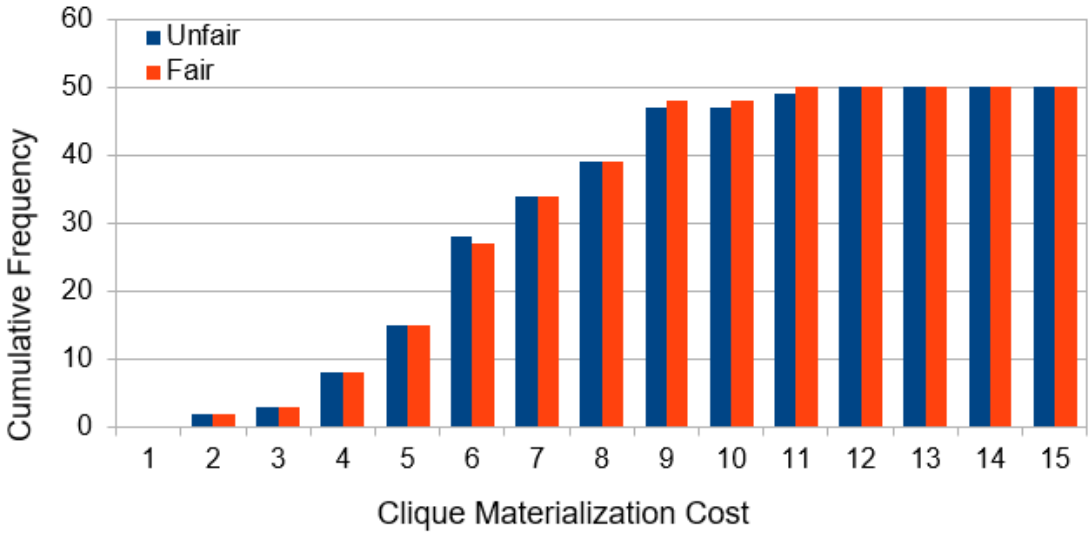


Figure 4.2: Selecting pre-calculated potential clusters: Cumulative frequency considering 110 available physical Things (mashups managed by the client).

$$OF^{\text{unfair}} : Minimize \sum_{r \in \mathcal{R}} \sum_{i \in \mathcal{T}^{\text{P}}} c_r^i \times \delta_r^i \tag{4.7}$$

Discussion

Plots in Figures 4.1 and 4.2 show the cumulative frequency values obtained for the fair and unfair objective functions for 90 and 110 physical Things respectively. The results obtained for 100 physical Things were similar (no

difference between fair and unfair approaches exists), and therefore are not shown. From plot at Figure 4.1, it is possible to observe that the fair approach reaches a cumulative frequency of 50 at clique materialization cost of 12. That is, the fair approach avoids 11 and 12 costs. Since these cost reflect the gap between the requirements of requests and physical Things, lower gaps allow physical Things with features that are above what is requested to be left for future requests and/or backup. This makes applications more resilient. Regarding plot at Figure 4.2, a similar behaviour, although not so foreshadowed, can be observed.

Both fair and unfair approaches have as a basis the cost vectors $\mathbf{c}_n$, which give the materialization cost based on maximum cliques. This reduces the search space (and, therefore, execution time) but eliminates potential solutions including cliques of smaller size. When increasing the population of cliques, better results can be obtained.

To conclude, a fair resource assignment when compared with the unfair one, minimizes the highest gap/cost. This allows physical Things with features that are above what is requested to be left for future requests and/or backup, meaning that applications have a lower probability of no finding available devices (devices fulfilling the requested property requirements). Algorithms to find populations of cliques with more potential and, therefore, better resource allocation are detailed in the next section.

4.3 Non Pre-calculated Clusters

4.3.1 Definitions and Assumptions

Definition 4 (Physical Thing)**.** *A sensor detecting events/changes, or an actuator receiving commands for the control of a mechanism. The model of a physical Thing i includes all properties necessary to describe it, denoted by $\mathcal{P}_i$, and all its functionalities, denoted by $\mathcal{F}_i$. That is, $\mathcal{P}_i = \{p : p \in \mathcal{P}\}$ and $\mathcal{F}_i = \{f : f \in \mathcal{F}\}$, where $\mathcal{P}$ is the overall set of properties (e.g., sensing range, communication facility, location), and $\mathcal{F}$ the overall set of functionalities (e.g., image sensor), from all devices registered at the cloud.*

Properties and functionalities, at $\mathcal{P}$ and $\mathcal{F}$, are assumed to be semantic-based. Thus, specific vocabularies are used when naming properties and functionalities (see [10]). It is also assumed that each property $p_i \in \mathcal{P}_i$ has a "subject-predicate-object" (or RDF triple) description associated with it (e.g., cameraResolution hasValue 12.1MP), which is denoted by $spo(p_i)$. The set of all registered physical Things is denoted by $\mathcal{T}^{\mathrm{P}}$, and it is assumed that providers voluntarily register/unregister physical Things to/from the cloud.

Definition 5 (Virtual Thing)**.** *Entity used for the mapping of multiple consumers into physical Things, having a virtual workspace associated it. A virtual Thing j can be materialized through one or more concerted physical Things, denoted by $\mathcal{M}_j$, $\mathcal{M}_j \subset \mathcal{T}^P$. Therefore[2], $f_j \triangleq \cup_{i \in \mathcal{M}_j} \mathcal{F}_i$ and $\mathcal{P}_j = \cup_{i \in \mathcal{M}_j} \mathcal{P}_i$. A virtual Thing materialization must fulfill the requirements of all its consumers.*

In other words, a virtual Thing has in background one or more physical Things working together to provide the requested functionality, producing data that reaches the cloud using standard communication. The set of virtual Things created by the cloud is denoted by $\mathcal{T}^{\mathrm{V}}$.

Clients/applications specify Thing requests (to the cloud) using web templates. Each request r has a functionality requirement and a set of minimum/maximum property requirements, denoted by $\bar{f}_r$ and $\bar{\mathcal{P}}_r$, respectively. The functionality and minimum/maximum property requirements are also semantic-based, each $p_r \in \bar{\mathcal{P}}_r$ having a "subject-predicate-object" description of

[2]The symbol $\triangleq$ means equal by definition, in our case logically/semantically equivalent.

the condition/requirement that is being defined (e.g., cameraResolution greaterThan 12.1MP; frequencySampling equalTo 10s), denoted by $spo(p_r)$. The overall population of requests (from all clients/applications) is denoted by $\mathcal{R}$.

Virtual Things, to be created at the cloud, are the ones to be materialized into physical Things. Then, each $r \in \mathcal{R}$ (element of mashup being managed at the client side) must be binded to a single virtual Thing, while a virtual Thing can be binded to multiple elements (with same functionality and compatible property requirements). With such approach, data generated by a virtual Thing can be consumed by multiple applications, reducing data collection/storage and increasing the usefulness of data. The right set of virtual Things to be created at the cloud, their bindings to requests and their materialization into physical Things should be determined while using resources efficiently, which is discussed next.

The goal of cloud virtualization is for users to remain unaware of physical devices involved in the process. This way, physical Things can be dynamically allocated to virtual Things used by applications. The client ends up having no deployment and maintenance costs, while having an on-demand fault tolerant service because virtual Things can always use other available physical Things. Clients would not be aware of such change due to virtualization.

4.3.2 Resource Allocation Mathematical Model

Let us assume a particular partition of $\mathcal{R}$ (population of requests), denoted by $\eta^i = \{\mathcal{R}^i_1, \mathcal{R}^i_2, ...\}$, where all elements in a $\mathcal{R}^i_j$ have the same functionality requirement. A virtual Thing $k \in \mathcal{T}^{\mathrm{V}}$ binded to $\mathcal{R}^i_j$ must provide the requested functionality, which is the same for all requests in $\mathcal{R}^i_j$. The following allocation function $f : \eta^i \rightarrow \mathcal{T}^{\mathrm{V}}$ can be defined:

$$f(\mathcal{R}^i_j) = \{\exists! k \in \mathcal{T}^{\mathrm{V}} : \bar{f}_k = f_r, \forall r \in \mathcal{R}^i_j\}. \tag{4.8}$$

One or more physical Things materialize one virtual Thing. Assuming $\tau^i = \{\mathcal{T}^{\mathrm{P},i}_1, \mathcal{T}^{\mathrm{P},i}_2, ...\}$ to be a specific partition of $\mathcal{T}^{\mathrm{P}}$, each $\mathcal{T}^{\mathrm{P},i}_j$ making sense from a functional point of view, the function $g : \tau^i \rightarrow \mathcal{T}^{\mathrm{V}}$ is defined for virtual Thing materialization:

$$g(\mathcal{T}_j^{\mathrm{P},i}) = \{\exists! k \in \mathcal{T}^{\mathrm{V}} : f_k \triangleq \cup_{l \in \mathcal{T}_j^{\mathrm{P},i}} \mathcal{F}_l\}. \tag{4.9}$$

This states that a virtual Thing $k \in \mathcal{T}^{\mathrm{V}}$ is materialized by $\mathcal{T}_j^{\mathrm{P},i}$, including one or more physical Things, if they are functionally similar.

Different partitions, and allocations done by f and g, have different impacts on the use of resources (cloud and physical Things) and provide different accomplishment levels for property requirements (more or less tight). Therefore, the best partitions should be determined. Let us assumed that η^{U} is the universe set including all feasible partitions of requests, $\mathcal{R}$. That is, $\eta^{\mathrm{U}} = \{\eta^1, \eta^2, ..., \eta^{|\eta^{\mathrm{U}}|}\}$ and $\eta^i = \{\mathcal{R}_1^i, \mathcal{R}_2^i, ..., \mathcal{R}_{|\mathcal{T}^{\mathrm{V}}|}^i\}$, $\forall i \in \{1, ..., |\eta^{\mathrm{U}}|\}$. Assume also τ^{U} to be the universe set including all feasible partitions of physical Things, $\mathcal{T}^{\mathrm{P}}$. That is, $\tau^{\mathrm{U}} = \{\tau^1, \tau^2, ..., \tau^{|\tau^{\mathrm{U}}|}\}$ and $\tau^i = \{\mathcal{T}_1^{\mathrm{P},i}, \mathcal{T}_2^{\mathrm{P},i}, ..., \mathcal{T}_{|\mathcal{T}^{\mathrm{V}}|}^{\mathrm{P},i}\}$, $\forall i \in \{1, ..., |\tau^{\mathrm{U}}|\}$. The impact of f and g allocations, regarding to the gap between requirements and properties of physical Things, can be described by the following cost function $h : \eta^{\mathrm{U}} \times \tau^{\mathrm{U}} \rightarrow \Re^+$:

$$h(\eta^i, \tau^j) = \sum_{\{\mathcal{R}_k^i \in \eta^i\}} \sum_{\{p \in \chi\}} min_{r \in \mathcal{R}_k^i} \{\Delta_{r,p}^{\mathrm{GAP}}(f(\mathcal{R}_k^i), \tau^j)\}, \tag{4.10}$$

where $\chi = \cup_{r \in \mathcal{R}_k^i} \bar{\mathcal{P}}_r$ includes all minimum/maximum property requirements from requests in $\mathcal{R}_k^i$. For each property $p \in \chi$, the min is used to capture the lowest gap between requirements and physical Things regarding property p (e.g., if two requests in $\mathcal{R}_k^i$ request for 12.1MP and 24.2MP camera resolutions, respectively, and the materialization of $\mathcal{R}_k^i$'s virtual Thing is a physical Thing providing 48.4MP, then the 24.2 to 48.4MP gap is the request-supply gap to be considered; the other request is considered to be fulfilled). Note that, $f(\mathcal{R}_k^i)$ returns the virtual Thing assigned to $\mathcal{R}_k^i$. Since multiple physical Things can be associated with a virtual Thing materialization, the $\Delta_{r,p}^{\mathrm{GAP}}$ at expression (4.10) must be defined by:

$$\Delta_{r,p}^{\mathrm{GAP}}(l, \tau^j) = \begin{cases} max_{t \in \mathcal{T}_k^{\mathrm{P},j} : \exists p_t = p, p_t \in \mathcal{P}_t} \{\Delta^{\mathrm{GAP}}(spo(p), spo(p_t))\}, \\ \text{, if } \exists \mathcal{T}_k^{\mathrm{P},j} \in \tau^j : g(\mathcal{T}_k^{\mathrm{P},j}) = l \\ \\ \infty, \text{otherwise} \end{cases} \tag{4.11}$$

where Δ^{GAP} provides the gap between the property requirement and property value at one of the physical Things enrolled in materialization. Since there might be more than one physical Thing, from all physical Things enrolled in a specific materialization, including a certain property then max is used to capture the highest gap value, avoiding virtual Thing materialization from having physical Things with property values far above the requirements. The best resource allocation will be:

$$(\eta^i, \tau^j)^* = argmin_{\eta^i \in \eta^{\text{U}}, \tau^j \in \tau^{\text{U}}} \{h(\eta^i, \tau^j)\}. \tag{4.12}$$

The previously mentioned gaps can be determined using SPARQL which is a language designed to query data across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. A variety of SPARQL processors are available for running queries against both local and remote data [2].

4.3.3 Resource Allocation Algorithm

Based on the previous model, a resource allocation algorithm is proposed in Algorithm 1. This is based on the following assumptions:

- As physical Things are registered at the cloud, possible materializations are computed using SPARQL. Therefore, there is a pool of materializations for a functionality f, denoted by $\mathcal{M}(f)$, each materialization involving one or more physical Things. A physical Thing may be at multiple pools.
- As consumer/application requests are inserted at the cloud, an auxiliary graph $\mathcal{G}(\mathcal{R}, \mathcal{L})$ is updated. The $\mathcal{R}$ includes all requests, while $\mathcal{L}$ denotes a set of links. A link between r_i and $r_j \in \mathcal{R}$ exists if: i) requests have the same functionality requirement; ii) property requirements are compatible. SPARQL is used to determine compatibility.

The first step initializes cost vectors, one per resource request r, denoted by $\mathbf{c}_r$. Since different materializations may exist, resulting into different materializations costs, these vectors have size $|\mathcal{M}(f_r)|$. The set of other requests joining r in a specific materialization (partition), to which a specific cost is associated with, will be stored in $\mathbf{v}_r$. When searching for feasible solutions, clique subgraphs are extracted from $\mathcal{G}(\mathcal{R}, \mathcal{L})$ and their possibility of materialization is analysed. Regarding the last step, where virtual Things are built

based on materialization cost vectors, different selection criteria have been considered:

- Cheapest materialization cost (CMC) first;
- Cheapest materialization, of request with less materialization choices (LMC), first;
- Cheapest materialization, of request with highest cost variance (HCV), first.

The reasoning behind LMC is that more materializations might be possible if critical requests are processed first. Regarding HCV, the reasoning is that a late selection of requests with highest cost variance might result in materializations with higher cost. The impact of these choices have been analysed and compared in the following section.

4.3.4 Performance Analysis

Scenario Setup

To carry out evaluation a pool of functionalities was created, each with its own pool of properties. Based on these, physical Things and requests were generated as follows:

- Physical Things have a randomly generated functionality with 50% of its properties (extrated from corresponding pool).
- The functionality required by each request is randomly selected from the pool of functionalities, together with 50% of its properties. Each (r_i, r_j) pair, $r_i, r_j \in \mathcal{R}$, sharing the same functionality requirement is assumed to be compatible with probability δ (see Section 4.3.3).
- The cost gap between a property requirement and a supplied device property is randomly selected from $\{\Delta_1, ..., \Delta_5\}$, where Δ_1 is the lowest cost and Δ_5 is the highest (moderate and extreme cost levels).

Table 4.1 shows the parameter values adopted for the performance evaluation of CMC, LMC and HCV strategies. The following section discusses results on the total resource allocation cost and on the total number of materializations (virtual Things).

Algorithm 1: Resource allocation heuristic (mashups managed by client).

```
/* Initialization step */;
for each r ∈ R do
    Create cost vector c_r of size |M(f_r)|;
    Create aggregation vector v_r of size |M(f_r)|;
    for each m ∈ M(f_r) do
        /* Initialize cost of possible materialization */;
        c_r(m) = ∞;
        /* Requests joining r in possible materialization*/;
        v_r(m) = ∅;
    end
end
/* Searching for feasible solutions */;
for each r_i ∈ R do
    /* pick clique subgraphs including r_i*/;
    R̄ = {Z ⊆ R : r_i ∈ Z ∧ (r_j, r_k) ∈ L, ∀r_j, r_k ∈ Z};
    /* pick maximum clique for which there is at least one feasible
       materialization */;
    R̄^MAX = argmax_{Z∈R̄:∃feasible m∈M(f_{n_i})}{ω(Z)};
    for each r_j ∈ R̄^MAX do
        /* for each possible materialization of f_{r_i} */;
        for each m ∈ M(f_{r_i}) do
            /* see best materialization cost for r_j*/;
            cost =internal sum of Eq. (5.3), using R^i_k = R̄^MAX;
            if c_{r_j}(m) > cost then
                c_{r_j}(m) = cost;
                v_{r_j}(m) = R̄^MAX;
            end
        end
    end
end
/* choose best resource allocations */;
Build virtual Things based on materialization cost vectors until all requests
  are fulfilled or no more devices exist;
```

Table 4.1: Simulation setup to evaluate CMC, LMC and HCV strategies (mashups managed by the client).

Functionality pool size	10
Avg size of property pools	10
Total number of devices	80
Device's properties (from pool)	50%
Requests's properties (from pool)	50%
δ	0.5
$\{\Delta_1, ..., \Delta_5\}$	$\{1, ..., 5\}$

Analysis of Results

The plot in Figure 4.3 shows the total resource allocation cost for CMC, LMC and HCV strategies. Results show that CMC and LCM strategies present

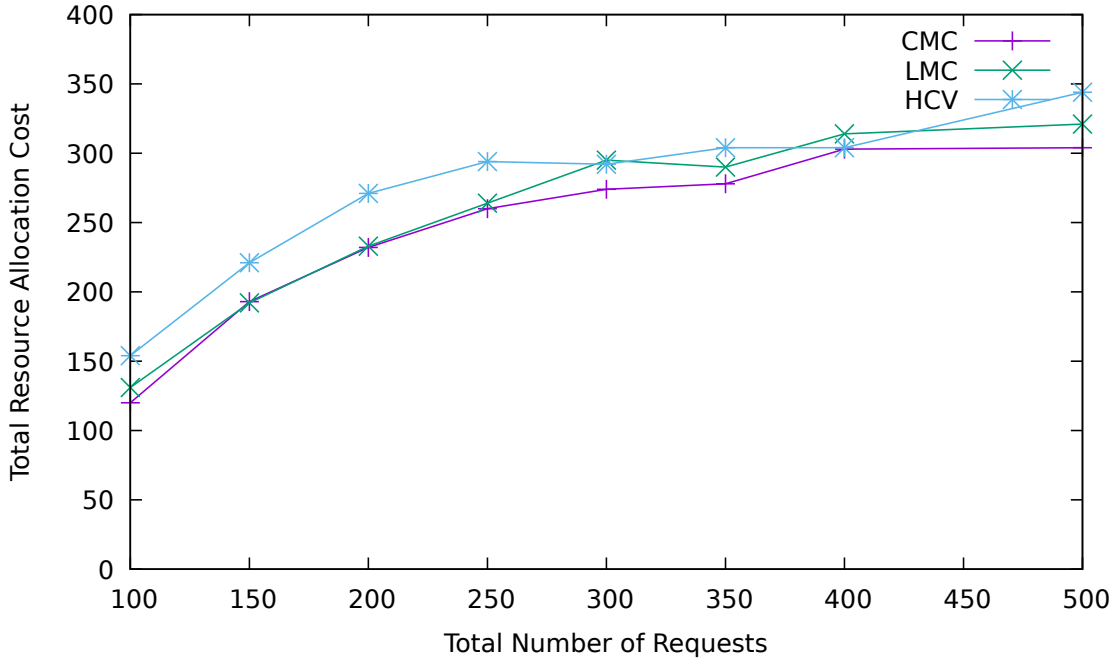


Figure 4.3: Building clusters: total resource allocation cost obtained by CMC, LCM and HCV (mashups managed by the client).

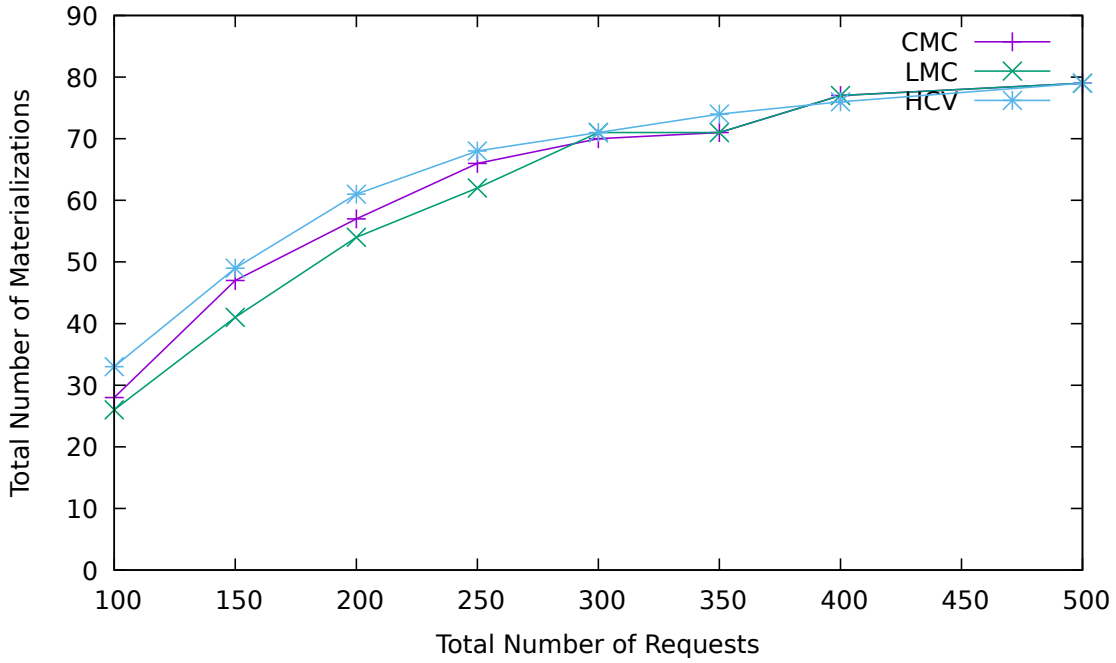


Figure 4.4: Building clusters: total number of materializations (virtual Things) obtained by CMC, LCM and HCV (mashups managed by the client).

the lowest overall resource allocation costs, meaning that the request-supply gap is lower than in HCV. That is, devices with properties closer to requests are being used for materialization, releasing devices with capabilities above

what is required for future requests. The relatively high resource allocation costs presented by strategy HCV means that choosing first the requests with highest cost variance leads to an increase in the overall materialization cost. Relatively high costs may end up being selected.

The plot in Figure 4.4 shows the total number of materializations for CMC, LMC and HCV strategies. This corresponds to the number of virtual Things or devices under utilization. From CMC and LMC strategies, the one requiring less devices is LMC. That is, the LMC is the most effective strategy since it requires less devices and assigns devices more adequate to the requirements of requests. Strategy HCV presents a relatively higher number of materializations, meaning that more virtual spaces (virtual Things) and devices are being used for a specific set of requests.

In summary, choosing first the requests with highest cost variance, which means that materialization cost could increase more if not treated first, does not improve by itself the overall materialization cost because relatively high costs may end up being selected. This approach also leads to the use of more devices, avoiding their availability for future requests. Therefore, this criteria should not be considered alone when searching for high quality allocation solutions, and must integrate with other parameters. Among the strategies under analysis, the LMC seems to be the most effective since it requires less devices and assigns devices more adequate to the requirements of requests.

4.3.5 Conclusions

In this chapter a resource allocation model for Se-aaS business models is addressed. The model fits multiple emerging IoT Se-aaS business models, like the ones supporting multi-sensing applications and/or integrating data from multiple domains, allowing for the orchestration of both sensor and cloud resources to face client requests. Results show that the model allows the implementation of multiple strategies, among which LMC seems to be the best choice. This strategy leads to the allocation of less devices, while the most adequate devices for consumer/application needs are also selected. This adequacy is measured through a semantics-based resource allocation cost.

Multiple strategies to select and build virtual Things were presented in this chapter. These assume that mashups are being managed at the client side. The next chapter assumes mashups being managed at the cloud, and the

previous mathematical model and algorithms are adapted for this purpose. That is, bindings between mashups (at the cloud) and client applications are assumed. A new mathematical optimization model and an extra heuristic are proposed to improve the results obtained by the just mentioned extended model.

CHAPTER 5

Resource Allocation in Sensing as-a-Service: Clouds Managing Mashups

5.1 Introduction

The cloud based Se-aaS model, for data and sensors from multiple suppliers to be shared by multiple clients, can bring benefits to all players in the IoT ecosystem [41]. Usually, software components (at the client) have bindings to virtual sensors managed in the cloud. The workflow generated by wiring together virtual sensors (mashups), actuators and services from different web sources, is traditionally managed at the client side. Mashup management at the client side results, however, into significant delays because of the permanent flow of data between clients and cloud.

When software components are able to bind to mashups managed at the cloud, the previously mentioned delays can be avoided. Events are processed and actuations can be triggered, according to the predefined workflow of the mashup, by the cloud which delivers just the final data to the consumer/client application. Thus, the whole mashup, or just parts of it, may be consumed by multiple applications/clients. This chapter proposes models and algorithms considering mashups managed at the cloud.

Contributions

The contributions presented on this chapter are:

- Extension of the mathematical model developed at the previous chapter, which finds the best clusters when mashups are managed at the client side, for the case of mashups managed in the cloud. The three heuristic approaches are also extended for this purpose.

- A new mathematical programming optimization model is developed that is able to obtain the optimal solution in resource allocation.
- Based on the knowledge gained with the development of the mathematical programming optimization model, a new heuristic is proposed. This is able to improve the results obtained by the previous heuristic approaches.

These contributions were published in:

- J. Guerreiro, L. Rodrigues and N. Correia, “Resource Allocation Model for Sensor Clouds under the Sensing as a Service Paradigm”, Computers Vol. 8, No. 1 (2019), [21].
- J. Guerreiro, L. Rodrigues and N. Correia, “Allocation of Resources in SeaaS Clouds Managing Virtual Sensor Mashups”, revisions of reviewers being done to *IEEE Transactions on Network and Service Management*, [17].

5.2 Model for Sensor Clouds Managing Mashups

The model proposed next is adequate for many emerging IoT Se-aaS business models, like the ones supporting multi-sensing applications, mashups of Things managed at the cloud, and/or integration of data from multiple domains, allowing for a more efficient orchestration of both sensor and cloud resources to face client requests.

5.2.1 Definitions and Assumptions

Definition 6 (Physical Thing)**.** *A sensor detecting events/changes, or an actuator receiving commands for the control of a mechanism. The model of a physical Thing i includes all properties necessary to describe it, denoted by $\mathcal{P}_i$, and all its functionalities, denoted by $\mathcal{F}_i$. That is, $\mathcal{P}_i = \{p : p \in \mathcal{P}\}$ and $\mathcal{F}_i = \{f : f \in \mathcal{F}\}$, where $\mathcal{P}$ is the overall set of properties (e.g., sensing range, communication facility, energy consumption, location), and $\mathcal{F}$ is the overall set of functionalities (e.g., image sensor), from all devices registered in the cloud.*

It is assumed that properties and functionalities, at $\mathcal{P}$ and $\mathcal{F}$, are semantic-based. That is, specific vocabularies are used when naming properties and functionalities (see [10]). In addition, each property $p_i \in \mathcal{P}_i$ will have a "subject-predicate-object" description associated with it denoted by $spo(p_i)$. The set of all registered physical Things is denoted by $\mathcal{T}^{\mathrm{P}}$, and it is assumed that providers voluntarily register/deregister physical Things to/from the cloud.

Definition 7 (Virtual Thing)**.** *Entity used for the mapping of multiple mashup elements (consumers) to physical Things, having a virtual workspace associated it. A virtual Thing j can be materialized through one or more concerted physical Things, denoted by $\mathcal{M}_j$, $\mathcal{M}_j \subset \mathcal{T}^P$. Therefore, $f_j \triangleq \cup_{i \in \mathcal{M}_j} \mathcal{F}_i$ and $\mathcal{P}_j = \cup_{i \in \mathcal{M}_j} \mathcal{P}_i$. A virtual Thing materialization must fulfil the requirements of all its consumers.*

Thus, a virtual Thing can have in background one or multiple physical Things working together to provide the requested functionality, producing data that reaches the cloud using standard communication. The set of virtual Things created by the cloud is denoted by $\mathcal{T}^{\mathrm{V}}$.

The set of all consumer applications is denoted by $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_{|\mathcal{A}|}\}$, and these are assumed to be outside the cloud. An application $\mathcal{A}_i$ can have one or more independent components, denoted by $\mathcal{C}(\mathcal{A}_i) = \{\mathcal{C}_1^i, ..., \mathcal{C}_{|\mathcal{C}(\mathcal{A}_i)|}^i\}$, and each component $\mathcal{C}_j^i$ has a binding to a mashup in the cloud.

Definition 8 (Mashup)**.** *Workflow wiring together a set of elements/nodes. Each element n included in a mashup has a functionality requirement and a set of property conditions, denoted by $\bar{f}_n$ and $\bar{\mathcal{P}}_n$, respectively.*

That is, it is assumed that user application components have bindings to mashups stored in the cloud, each mashup including elements connected by a workflow (web templates can be used to draw mashups). The output of a mashup element can be input to another, while final mashup output data is sent to the corresponding application component. The functionality requested by a mashup element, and property conditions, are also semantic-based and each $p_n \in \bar{\mathcal{P}}_n$ has a "subject/predicate/object" description of the condition that is being defined, denoted by $spo(p_n)$. Thus, mashup elements are not physical Things, but, instead, nodes that specify requirements. The overall population of mashup elements (from all applications) is denoted by $\mathcal{N}$.

Virtual Things, to be created in the cloud, are the ones to be materialized into physical Things. Then, each mashup element $n \in \mathcal{N}$ must be mapped to a single virtual Thing, while a virtual Thing can be mapped to multiple mashup elements (with same functionality and compatible property requirements). With such approach, data generated by a virtual Thing can be consumed by multiple application mashup elements, reducing data collection/storage and increasing the usefulness of data. The right set of virtual Things to be created in the cloud, their mapping to mashup elements and their materialization onto physical Things should be determined while using resources efficiently, which is discussed in the following section.

5.2.2 Architecture with Embedded Mashups

The virtualization approach shown in Figure 2.4 allows sensors/data to be accessed and mashups to be built at the client side. As an example, an application may use data from VS1 and VS3 to decide on some actuation. However, if such workflows (wiring together VSs, actuators and services from various Web sources) were implemented in the cloud, some of the data would not have to travel to the client side. The cloud would ensure that events are processed and actuations are triggered, according to the predefined workflow of mashups, delivering just the final data of interest to the consumer/client application. The whole mashup, or parts of it, may also be consumed by multiple applications. This additional system functionality results in an additional mashup virtualization layer, as illustrated in Figure 5.1.

Managing mashups in the cloud brings new challenges when assigning resources (devices and cloud) to consumer needs. More specifically, mashups end up defining flow dependencies (see Figure 5.1), which will influence:

- Mapping between one or more mashup elements (defined by consumers) and a virtual Thing, for resource optimization.
- Mapping between virtual Things and physical Things (materialization onto devices).
- Placement of virtual Thing workspaces in the cloud.

Thus, after mapping a virtual Thing to one or more mashup elements, such virtual Thing ends up participating in multiple mashups. The just mentioned mappings should be done having some criteria in mind, like an efficient use of physical Things and cloud resources, a reduction of flows or delay between virtual Thing workspaces (imposed by mashups), which improves user's quality of experience and scalability. This approach fits many emerging IoT Se-aaS business models, like the ones supporting multi-sensing applications, mashups of Things, and/or integration of data from multiple domains. Note that the approach in Figure 2.4 will be a particular case of Figure 5.1 where mashups have a single element.

5.2.3 Resource Allocation Mathematical Model

Let us assume a particular partition of $\mathcal{N}$ (population of mashup elements), denoted by $\eta^i = \{\mathcal{N}_1^i, \mathcal{N}_2^i, ...\}$, where all elements in a $\mathcal{N}_j^i$ have the same functionality requirement. A virtual Thing $k \in \mathcal{T}^{\mathrm{V}}$ mapped to $\mathcal{N}_j^i$ must provide

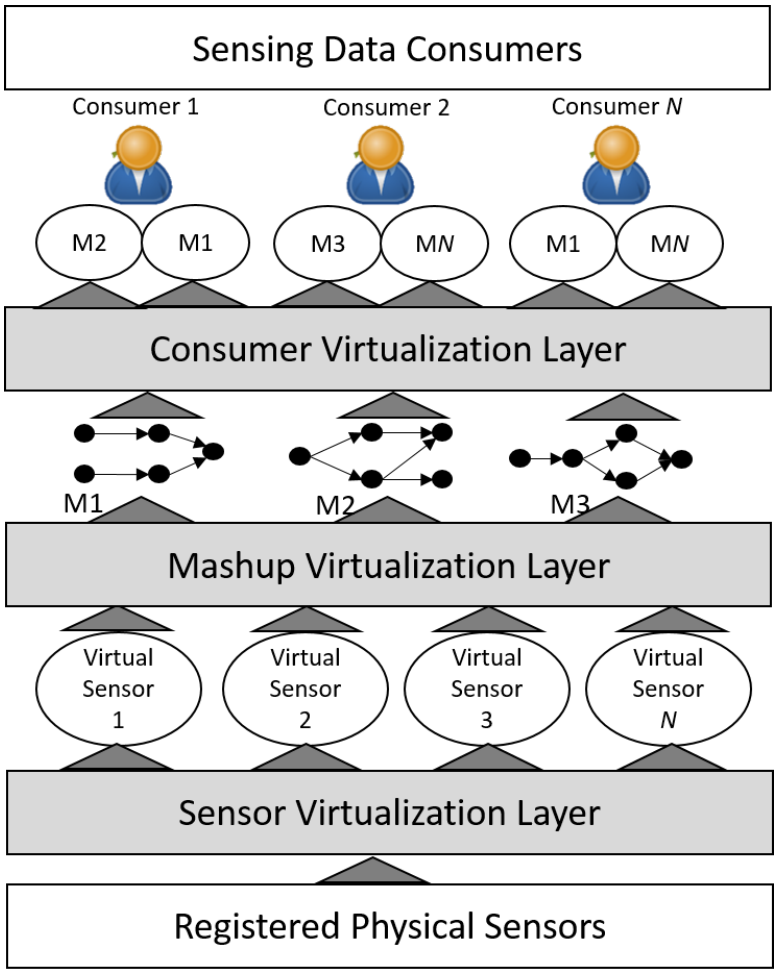


Figure 5.1: Virtualization layers in Se-aaS considering mashups embedded into the cloud.

the requested functionality, which is the same for all mashup elements in $\mathcal{N}_j^i$. The following allocation function $f : \eta^i \rightarrow \mathcal{T}^{\mathrm{V}}$ can be defined:

$$f(\mathcal{N}_j^i) = \{\exists! k \in \mathcal{T}^{\mathrm{V}} : \bar{f}_k = f_n, \forall n \in \mathcal{N}_j^i\}. \tag{5.1}$$

One or more physical Things materialize one virtual Thing. Assuming $\tau^i = \{\mathcal{T}_1^{\mathrm{P},i}, \mathcal{T}_2^{\mathrm{P},i}, ...\}$ to be a specific partition of $\mathcal{T}^{\mathrm{P}}$, each $\mathcal{T}_j^{\mathrm{P},i}$ making sense from a functional point of view, the function $g : \tau^i \rightarrow \mathcal{T}^{\mathrm{V}}$ is defined for virtual Thing materialization:

$$g(\mathcal{T}_j^{\mathrm{P},i}) = \{\exists! k \in \mathcal{T}^{\mathrm{V}} : f_k \triangleq \cup_{l \in \mathcal{T}_j^{\mathrm{P},i}} \mathcal{F}_l\}. \tag{5.2}$$

This states that a virtual Thing $k \in \mathcal{T}^{\mathrm{V}}$ is materialized by $\mathcal{T}_j^{\mathrm{P},i}$, including one or more physical Things, if they are functionally similar.

Different partitions, and allocations done by f and g, have different impacts on the use of resources (cloud and physical Things) and provide different accomplishment levels for property requirements (more or less tight). Therefore, the best partitions should be determined. Let us assumed that η^{U} is the universe set including all feasible partitions of mashup elements, $\mathcal{N}$. That is, $\eta^{\mathrm{U}} = \{\eta^1, \eta^2, ..., \eta^{|\eta^{\mathrm{U}}|}\}$ and $\eta^i = \{\mathcal{N}_1^i, \mathcal{N}_2^i, ..., \mathcal{N}_{|\mathcal{T}^{\mathrm{V}}|}^i\}$, $\forall i \in \{1, ..., |\eta^{\mathrm{U}}|\}$. Also assume that τ^{U} is the universe set including all feasible partitions of physical Things, $\mathcal{T}^{\mathrm{P}}$. That is, $\tau^{\mathrm{U}} = \{\tau^1, \tau^2, ..., \tau^{|\tau^{\mathrm{U}}|}\}$ and $\tau^i = \{\mathcal{T}_1^{\mathrm{P},i}, \mathcal{T}_2^{\mathrm{P},i}, ..., \mathcal{T}_{|\mathcal{T}^{\mathrm{V}}|}^{\mathrm{P},i}\}$, $\forall i \in \{1, ..., |\tau^{\mathrm{U}}|\}$. The impact of f and g allocations, regarding the gap between requirements and properties of physical Things, can be described by the following cost function $h : \eta^{\mathrm{U}} \times \tau^{\mathrm{U}} \rightarrow \Re^+$:

$$h(\eta^i, \tau^j) = \sum_{\{\mathcal{N}_k^i \in \eta^i\}} \sum_{\{p \in \chi\}} min_{n \in \mathcal{N}_k^i} \{\Delta_{n,p}^{\mathrm{GAP}}(f(\mathcal{N}_k^i), \tau^j)\}, \tag{5.3}$$

where $\chi = \cup_{n \in \mathcal{N}_k^i} \bar{\mathcal{P}}_n$ includes all properties, having conditions, from mashup elements in $\mathcal{N}_k^i$. For each property $p \in \chi$, the min is used to capture the lowest gap between requirements and physical Things regarding property p (e.g., if two elements in $\mathcal{N}_k^i$ request for 12.1 MP and 24.2 MP camera resolutions, respectively, and the materialization of $\mathcal{N}_k^i$'s virtual Thing is a physical Thing providing 48.4 MP, then the 24.2 to 48.4 MP gap is the request-supply gap to be considered; the other request is considered to be fulfilled). Note that $f(\mathcal{N}_k^i)$ returns the virtual Thing assigned to $\mathcal{N}_k^i$. Since multiple physical Things can be associated with a virtual Thing materialization, the $\Delta_{n,p}^{\mathrm{GAP}}$ at Equation (5.3) must be defined by:

$$\Delta_{n,p}^{\mathrm{GAP}}(l, \tau^j) = \begin{cases} max_{t \in \mathcal{T}_k^{\mathrm{P},j} : \exists p_t = p, p_t \in \mathcal{P}_t} \{\Delta^{\mathrm{GAP}}(spo(p), spo(p_t))\}, \\ , \text{if } \exists \mathcal{T}_k^{\mathrm{P},j} \in \tau^j : g(\mathcal{T}_k^{\mathrm{P},j}) = l, \\ \infty, \text{otherwise,} \end{cases} \tag{5.4}$$

where Δ^{GAP} provides the gap between the property requirement and property value at one of the physical Things enrolled in materialization. Multiple physical Things may include a property and, therefore, max is used to capture the highest gap value, in order to avoid virtual Thing materializations from having physical Things with property values far above the requirements. All the just mentioned gaps can be determined using SPARQL semantic query language [8, 9].

Having the previous definitions in mind, the best partitioning for $\mathcal{N}$ and $\mathcal{T}^{\mathrm{P}}$, determining which virtual Things should be built and their materialization, could be given by $argmin_{\eta^i \in \eta^{\mathrm{U}}, \tau^j \in \tau^{\mathrm{U}}}\{h(\eta^i, \tau^j)\}$. This would provide scalable solutions because the number of required virtual Things (and virtual workspaces) ends up being minimized (see Equation (5.3)). However, mashups define flows between their elements. This means that, after mapping partitions of $\mathcal{N}$ (mashup elements) into virtual Things, there will be flows between virtual workspaces of virtual Things. These flows must be taken into account so that scalability and QoE are not jeopardized due to overhead and delay in the cloud. Therefore, an additional cost function $h' : \eta^U \times \tau^U \rightarrow \Re^+$ is defined as:

$$h'(\eta^i, \tau^j) = \sum_{\{\mathcal{N}_k^i \in \eta^i\}} \sum_{\{\mathcal{T}_l^{\mathrm{P},j} \in \tau^j\}} TF^{\mathrm{P2V}}(\mathcal{T}_l^{\mathrm{P},j}, \mathcal{N}_k^i) + \sum_{\{\mathcal{N}_k^i \in \eta^i\}} \sum_{\{\mathcal{N}_{k'}^i \in \eta^i\}} TF^{\mathrm{V2V}}(\mathcal{N}_{k'}^i, \mathcal{N}_k^i) + \\ + \sum_{\{\mathcal{N}_k^i \in \eta^i\}} \sum_{\{\mathcal{A}_i \in \mathcal{A}\}} TF^{\mathrm{V2A}}(\mathcal{N}_k^i, \mathcal{A}_i) \quad (5.5)$$

where:

- TF^{P2V} is a physical-to-virtual (P2V) transfer cost associated with the flow of data from physical Things to virtual Thing's workspace in the cloud. This is zero if $f(\mathcal{N}_k^i) \neq g(\mathcal{T}_l^{\mathrm{P},j})$, meaning that $\mathcal{T}_l^{\mathrm{P},j}$ is not used in the materialization of $\mathcal{N}_k^i$'s virtual Thing;
- TF^{V2V} is a virtual-to-virtual (V2V) transfer cost associated with the flow of data between virtual Thing's workspaces of partitions $\mathcal{N}_{k'}^i$ and $\mathcal{N}_k^i$. This is zero if no flow between workspaces is required;
- TF^{V2A} is a virtual-to-application (V2A) transfer cost associated with flow of data from virtual Things' workspaces to user applications. This is zero if the application is supposed to consume such data.

Transfer costs may reflect the number of hops and/or processing needs at these hops, meaning that it is dependent on the placement of virtual workspaces in the cloud. A CSP, which will be denoted by $\mathcal{S}$, often includes a set of distributed networks, that interconnect to provide services, and are usually organized in order to better serve certain regions. Therefore, a CSP is defined by $\mathcal{S} = \{\mathcal{S}_1, ...\mathcal{S}_{|\mathcal{S}|}\}$, where $\mathcal{S}_i$ includes a set of computing resources that can host virtual workspaces.

Finally, the best resource allocation, or partitioning for $\mathcal{N}$ and $\mathcal{T}^{\mathrm{P}}$, is defined by:

$$(\eta^i, \tau^j)^* = argmin_{\eta^i \in \eta^{\mathrm{U}}, \tau^j \in \tau^{\mathrm{U}}} \{\alpha \times \hat{h}(\eta^i, \tau^j) + \beta \times \hat{h}'(\eta^i, \tau^j)\}, \tag{5.6}$$

where α and β are weights, $\alpha + \beta = 1$, defining the relative importance of normalized (Normalization formula: $\frac{x - x^{\min}}{x^{\max} - x^{\min}}$) costs, $\hat{h}$ and $\hat{h}'$.

5.2.4 Resource Allocation Algorithm

Based on the previous model, a resource allocation algorithm is proposed next. It is assumed that:

- As physical Things are registered in the cloud, a pool of possible materializations is computed for each functionality, denoted by $\mathcal{M}(f)$, using SPARQL. A materialization may involve one or more registered physical Things, and a physical Thing may be at multiple pools.
- As application mashups are inserted in the cloud, an auxiliary graph $\mathcal{G}(\mathcal{N}, \mathcal{L}, \mathcal{L}')$ is updated. The $\mathcal{N}$ includes all mashup elements, $\mathcal{L}$ are the links denoting a flow between two elements of a mashup, and $\mathcal{L}'$ are compatibility links between two elements from any mashup. That is, a link between n_i and $n_j \in \mathcal{N}$ exists in $\mathcal{L}'$ if: (i) nodes have the same functionality requirement; and (ii) property requirements are compatible (SPARQL is used to determine compatibility).

The resource allocation algorithm is described in Algorithm 2. The initialization step builds a partition for each mashup element, generates random places in CSP resources for them, and assigns an infinite cost. This has to be done in a per materialization basis because different materializations involve different physical Things, generating different costs, and some materializations may not even be feasible due to mashup element property conditions. For random placement of partition's virtual Thing workspace, a uniform distribution is used for load balancing to be obtained in the long term. The second step improves this initial solution by analysing cliques in auxiliary graph $\mathcal{G}(\mathcal{N}, \mathcal{L}')$, and materialization possibilities.

For the last step, different selection criteria have been compared: i) cheapest materialization cost (CMC) first; ii) cheapest materialization, of mashup element (node in graph in $\mathcal{N}$) with fewer materialization choices (LMC), first;

Algorithm 2: Resource allocation heuristic (mashups managed by the cloud).

Input: $\mathcal{N}$, $\mathcal{L}$, $\mathcal{L}'$, $\mathcal{M}(f)\ \forall f \in \mathcal{F}$, $\mathcal{S}$, α, β
/ Initialization step */*
for *each* $n \in \mathcal{N}$ **do**
  Create vector of partitions $\mathbf{v}_n$ of size $|\mathcal{M}(f_n)|$
  Create vector of places $\mathbf{p}_n$ of size $|\mathcal{M}(f_n)|$
  Create vector of costs $\mathbf{c}_n$ of size $|\mathcal{M}(f_n)|$
  for *each* $m \in \mathcal{M}(f_n)$ **do**
    / Nodes joining n in partition, materialized by m */*
    $\mathbf{v}_n(m) = \emptyset$
    / Place for partition's virtual Thing workspace */*
    $\mathbf{p}_n(m) = \text{RANDOMSELECTION}(\mathcal{S})$
    / Materialization costs, initially set to infinity */*
    $\mathbf{c}_n^1(m) = \infty$
    $\mathbf{c}_n^2(m) = \infty$
  end
end
/ Improving feasible solution */*
for *each* $n_i \in \mathcal{N}$ **do**
  / Clique subgraphs in $\mathcal{G}(\mathcal{N}, \mathcal{L}')$ that include n_i*/*
  $\bar{\mathcal{N}} = \{\mathcal{Z} \subseteq \mathcal{N} : n_i \in \mathcal{Z} \wedge (n_j, n_k) \in \mathcal{L}', \forall n_j, n_k \in \mathcal{Z}\}$
  / Maximum clique for which there is at least one feasible materialization */*
  $\bar{\mathcal{N}}^{\max} = argmax_{\mathcal{Z} \in \bar{\mathcal{N}}: \exists feasible\ m \in \mathcal{M}(f_{n_i})}\{\omega(\mathcal{Z})\}$
  / Random place for $\bar{\mathcal{N}}^{\max}$'s workspace */*
  $pl = \text{RANDOMSELECTION}(\mathcal{S})$
  for *each* $n_j \in \bar{\mathcal{N}}^{max}$ **do**
    / for each possible materialization of f_{n_i} */*
    for *each* $m \in \mathcal{M}(f_{n_i})$ **do**
      / determine best materialization cost for n_j*/*
      $c^1 =$ Equation (5.3) considering $\mathcal{N}_k^i = \bar{\mathcal{N}}^{\max}$
      $c^2 =$ Equation (5.5) considering n_j's in/out flow
      if $\alpha \times \hat{\boldsymbol{c}}_{n_j}^1(m) + \beta \times \hat{\boldsymbol{c}}_{n_j}^2(m) > \alpha \times \hat{c}^1 + \beta \times \hat{c}^2$ **then**
        $\mathbf{p}_n(m) = pl$
        $\mathbf{v}_{n_j}(m) = \bar{\mathcal{N}}^{\max}$
        $\mathbf{c}_{n_j}^1(m) = c^1$
        $\mathbf{c}_{n_j}^2(m) = c^2$
      end
    end
  end
end
/ choose best resource allocations */*
Build virtual Things based on materialization cost vectors until all requests are fulfilled or no more devices exist

iii) cheapest materialization, of mashup element with highest cost variance (HCV), first.

5.2.5 Performance Analysis

Scenario Setup

To carry out evaluation, a pool of functionalities was created together with a pool of properties for each functionality. Based on these, physical Things and mashup elements were created as follows:

- Mashups were randomly generated using the algorithm in [39], which is suitable for the generation of sparse sensor-actuator networks. An average of 10 elements per mashup is defined.
- The functionality required by each mashup element is randomly selected from the pool of functionalities, together with 50% of its properties. Each pair $n_i, n_j \in \mathcal{N}$ sharing the same functionality requirement is assumed to be compatible with probability δ.
- A physical Thing has a functionality assigned to it, together with 50% of its properties (randomly extracted from corresponding pool).
- The gap between a property condition and device property is randomly selected from $\{\Delta_1, ..., \Delta_5\}$, where Δ_1 is the lowest cost and Δ_5 is the highest (moderate and extreme levels).

This information is used to generate random scenarios, from which results are extracted. Regarding the CSP network graph, this was randomly generated assuming $|\mathcal{S}| = 10$ (number of places with computing resources that can host virtual workspaces) and a network density (Network density is measured using $\frac{L}{N\times(N-1)}$, where L is the number of links and N is the number of nodes) of 0.25. Tests include α and β values equal to 0.25 or 0.75, $\alpha + \beta = 1$, so that the impact of component costs in Label (5.6) can be evaluated. Table 5.1 summarizes the adopted parameter values. All simulations were performed using C++ programming language.

Materializations and Fulfilled Mashup Elements

The plots in Figures 5.2–5.4 show how CMC, LMC and HCV strategies perform regarding the number of materializations (number of virtual Things materialized into physical devices), number of elements from mashups that have been fulfilled (mapped to a virtual Thing and, therefore, materialized into a physical device) and the average number of mashup elements per virtual Thing (average size of clique $\bar{\mathcal{N}}^{\max}$ at Algorithm 2). From such plots, it is possible to observe that the worst strategy is LMC that reaches the total

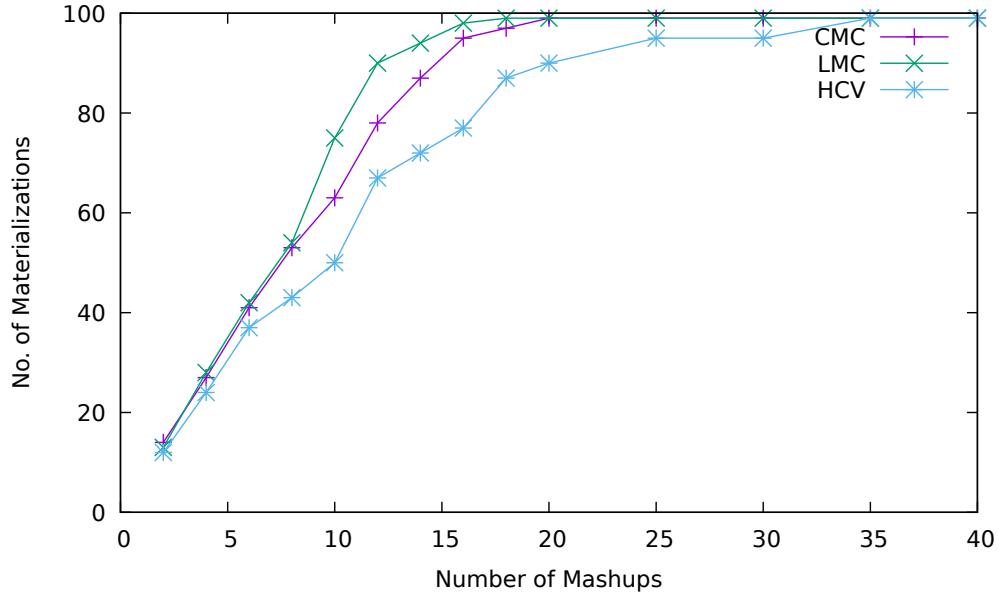
Table 5.1: Simulation setup to evaluate CMC, LMC and HCV strategies (mashups managed by the cloud).

Parameter	Value
Functionality pool size	10
Avg size of property pools	10
Total number of devices	100
Device's properties (from pool)	50%
Avg number of elements per mashup	10
Mashup element's properties (from pool)	50%
δ	0.5
$\{\Delta_1, ..., \Delta_5\}$	$\{1, ..., 5\}$
α, β	0.25 or 0.75; $\alpha + \beta = 1$
$\|\mathcal{S}\|$	10
CSP density	0.25

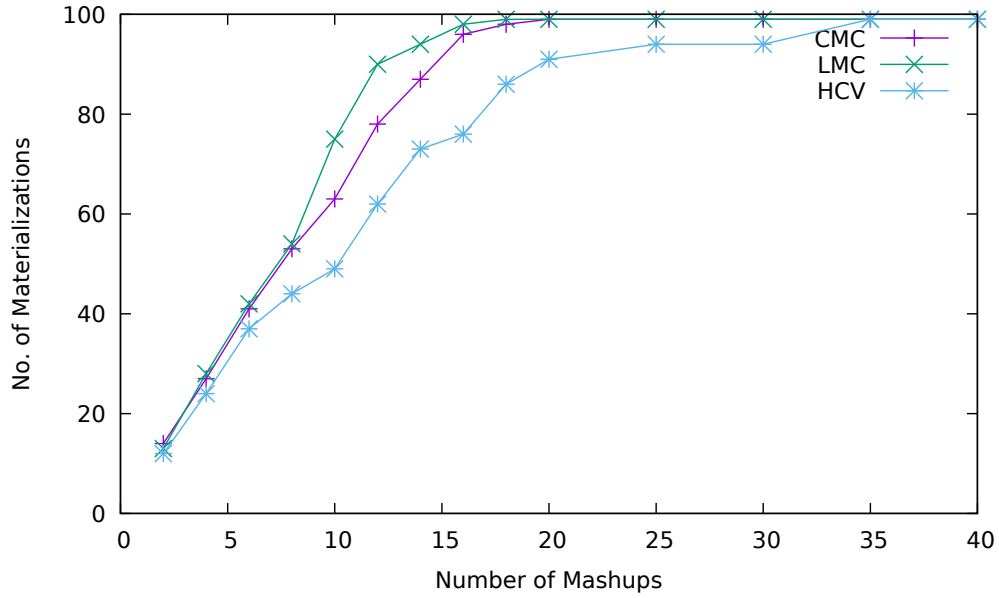
number of available devices more quickly while fulfilling fewer mashup elements than the other strategies. This is confirmed by the relatively low average number of mashup elements mapped to virtual Things (low aggregation level). The approach of LMC is to choose the cheapest materialization from mashup elements with fewer materialization choices, based on the assumption that more materializations would be possible if critical mashup elements were processed first. However, such mashup elements end up being the ones with more incompatible requirements (reason behind having fewer materialization choices), leading to a less efficient use of physical devices. That is, for a specific number of mashups, more devices are used for materialization of virtual Things having a low level of aggregation.

The best strategy is HCV that presents a higher average number of mashup elements mapped to virtual Things, when compared with the other strategies, and more fulfilled mashup elements. Such highest aggregation level leads to a more controlled use of available devices, which are not wasted with materialization of virtual Things having a low level of aggregation. The approach of HCV is to pick the cheapest materialization from mashup elements with highest cost variance. This is based on the assumption that their late selection could result in a materialization with high cost. It happens that a mashup element having high cost variance also means that such mashup element has more requirements that are compatible with others, leading to a higher level of aggregation. For this reason, HCV presents better results.

Note that, for a relatively low number of mashups, the difference between strategies, regarding the number of virtual Things materialized into devices



(a) $\alpha = 0.25$

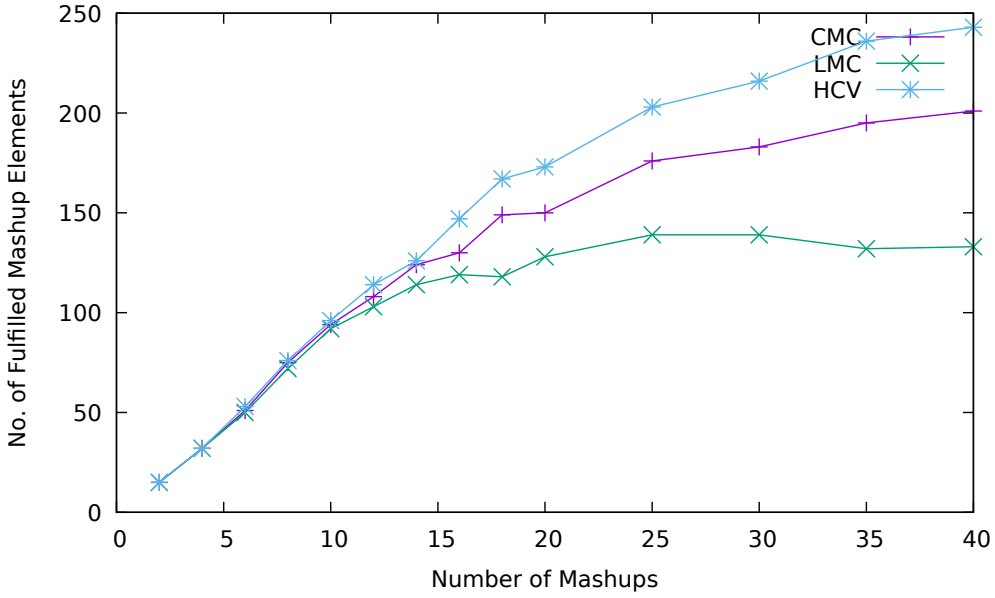


(b) $\alpha = 0.75$

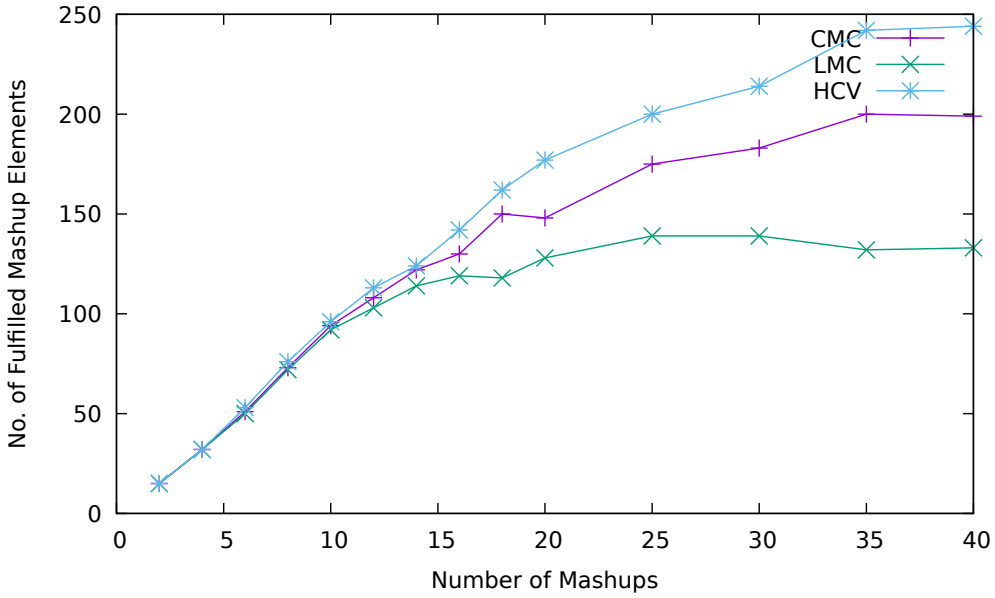
Figure 5.2: Number of virtual Things materialized into physical devices obtained by CMC, LMC and HCV strategies (mashups managed by the cloud).

and fulfilled mashup elements, is very low. This is because the population of mashup elements is small, not allowing a high level of aggregation. That is, it is more difficult to find mashup elements with compatible requirements, for these to be linked to the same virtual Thing.

Regarding the impact of changing α, which is the importance given to the cost associated with the gaps between the mashup element's properties and physical Thing's properties, it looks like this does not influence the number of



(a) $\alpha = 0.25$



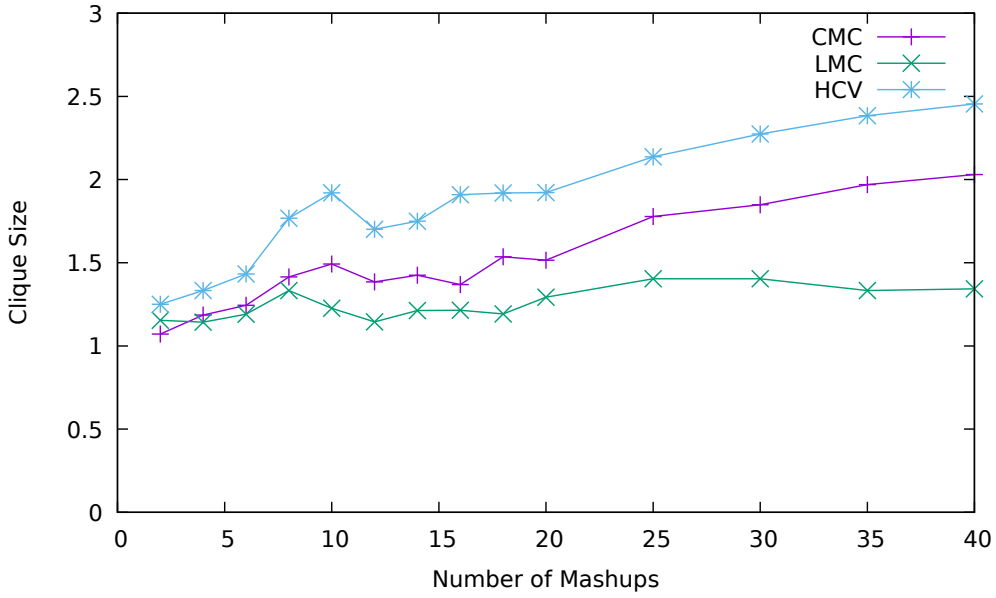
(b) $\alpha = 0.75$

Figure 5.3: Number of fulfilled mashup elements obtained by CMC, LMC and HCV strategies (mashups managed by the cloud).

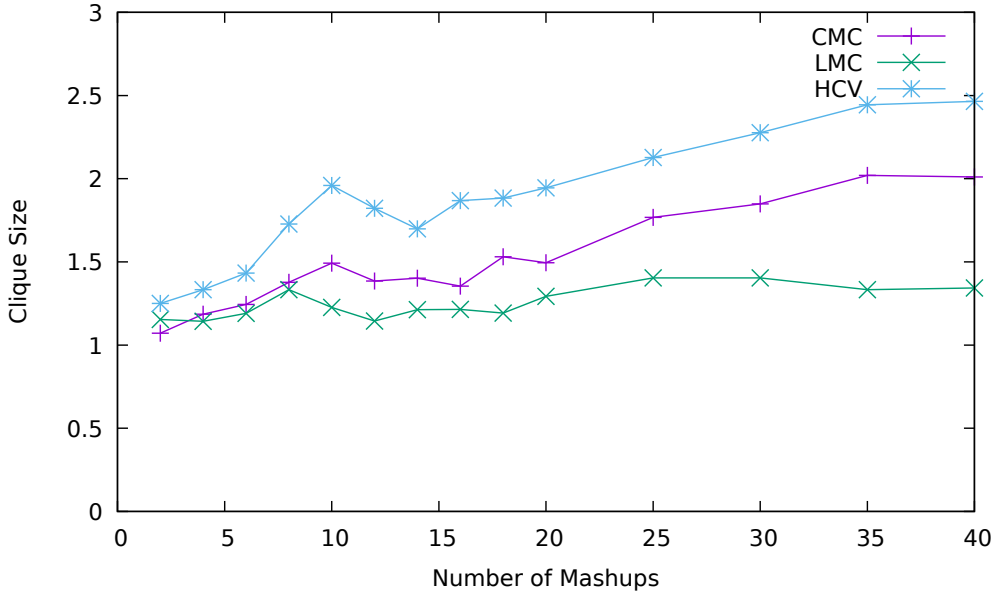
virtual Things materialized into devices and fulfilled mashup elements.

Cost

The plots in Figures 5.5, 5.6, 5.7 show the overall materialization cost resulting from CMC, LMC and HCV strategies, together with the two cost components associated with Equations (5.3) and (5.5), c^1 and c^2 in Algorithm 2, respectively. Plots show that, when the number of devices in use is not close



(a) $\alpha = 0.25$

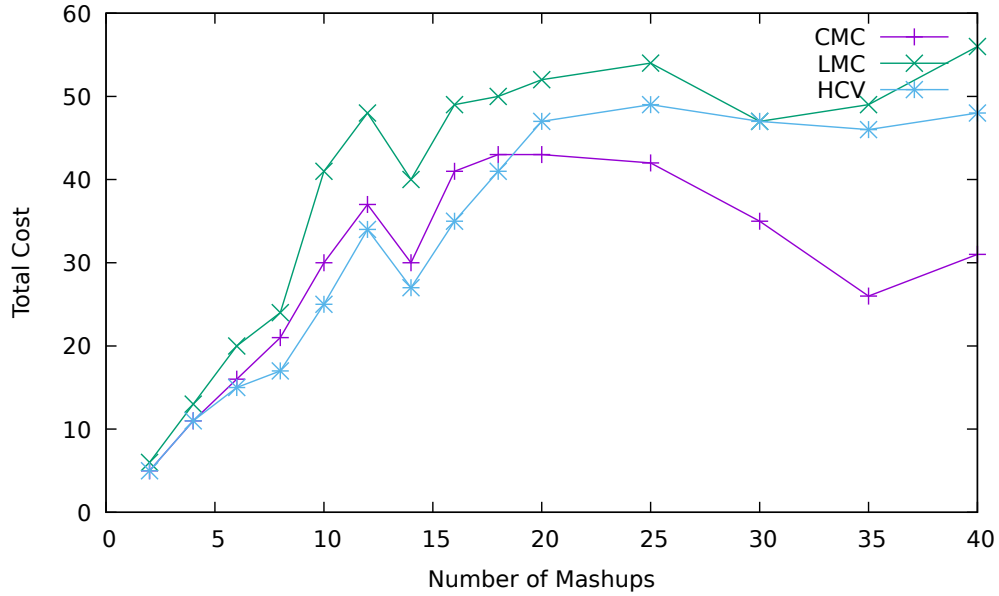


(b) $\alpha = 0.75$

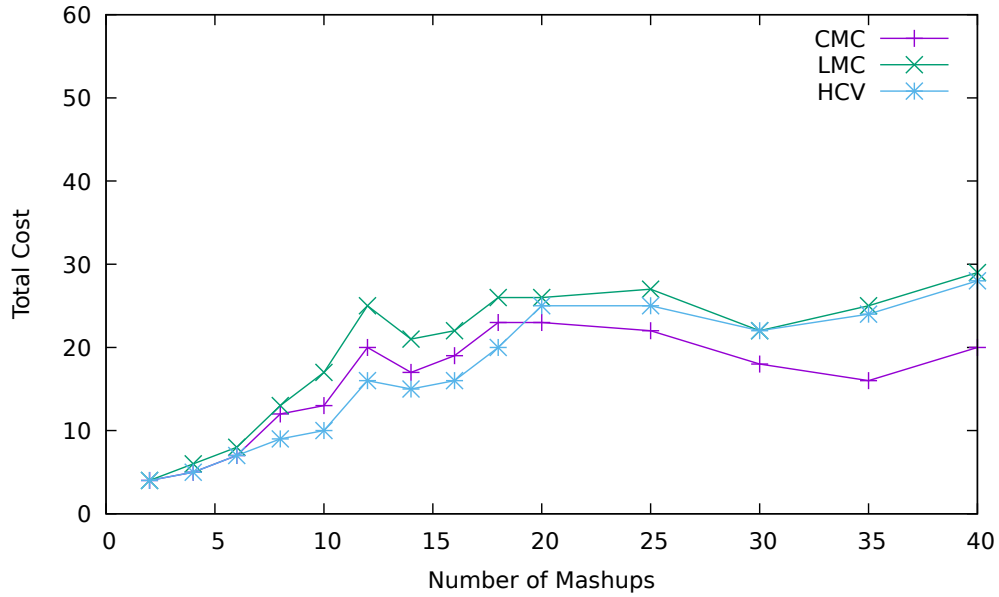
Figure 5.4: Average number of mashup elements per virtual Thing obtained by CMC, LMC and HCV strategies (mashups managed by the cloud).

to the limit, the highest cost is the one given by LMC because of its low aggregation level (low average number of mashup elements mapped to virtual Things). HCV ends up providing the best cost values because it makes more aggregations. More specifically, the min in Equation (5.3) captures the lowest gap between requirements of mashup elements (linked to a virtual Thing) and physical Things, reducing the overall cost.

After all the devices are in use; however, CMC is the one able to reduce the overall cost because it has the freedom to search for the cheapest cost,



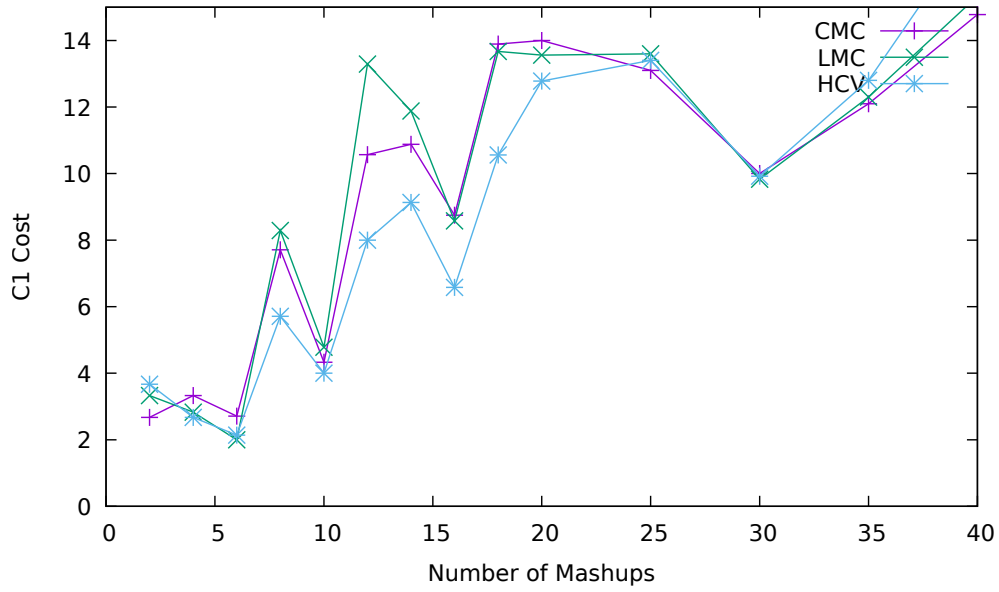
(a) Total cost using $\alpha = 0.25$



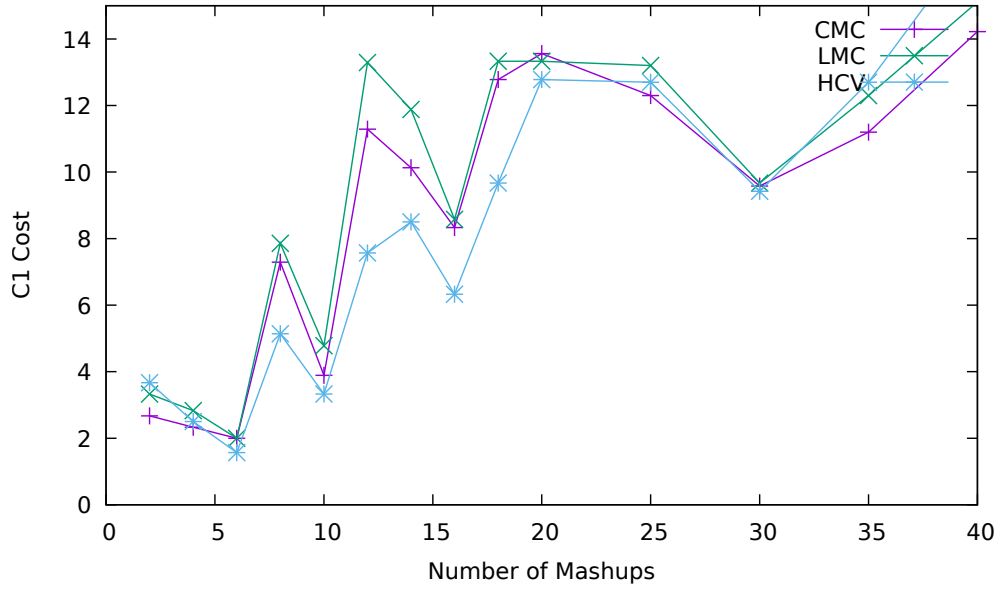
(b) Total cost using $\alpha = 0.75$

Figure 5.5: Total materialization cost obtained by CMC, LMC and HCV strategies (mashups managed by the cloud).

while the other strategies are conditioned in the search. LMC must pick the cheapest cost from mashup elements with fewer materialization choices, while HCV must pick the cheapest cost from mashup elements with highest cost variance. Although the population of mashup elements is greater, CMC does not improve its overall cost thanks to c^1 component (related with gaps between the mashup element's properties and physical Thing's properties) because this strategy has a low aggregation level. Instead, the reduction is achieved thanks to c^2 component, meaning that better placements for vir-



(a) c^1 using $\alpha = 0.25$

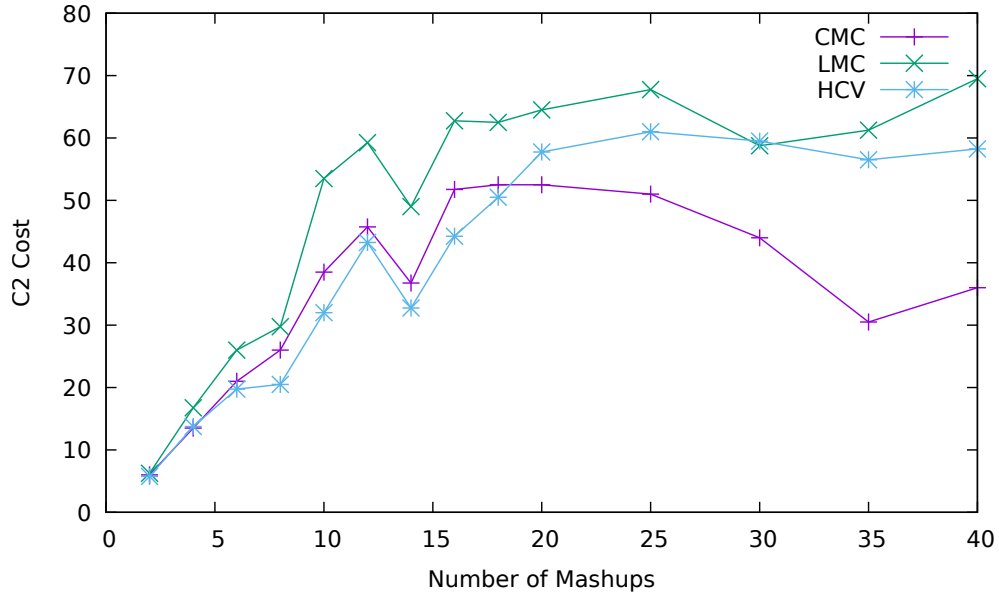


(b) c^1 using $\alpha = 0.75$

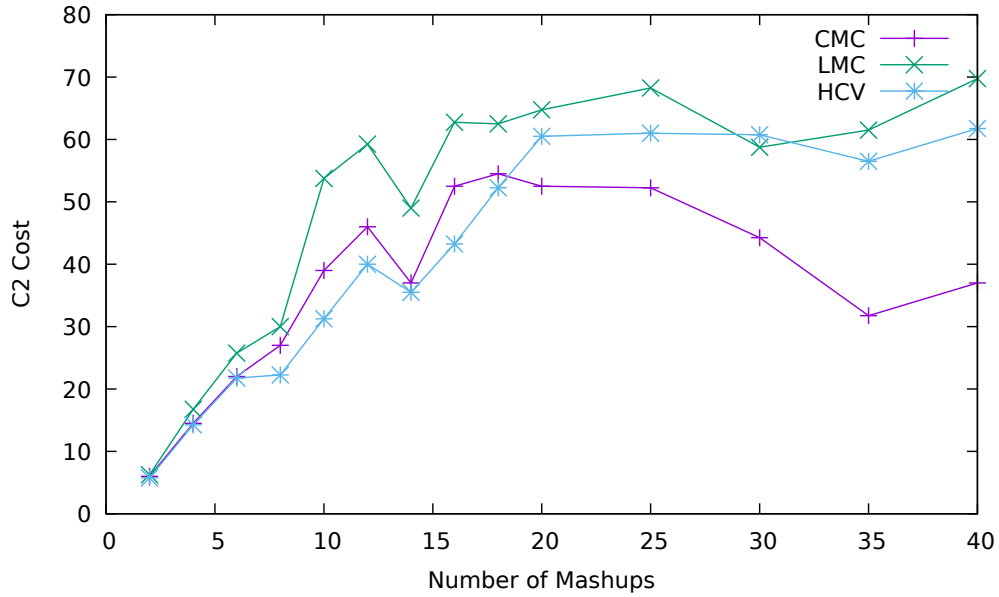
Figure 5.6: C1 component of materialization cost obtained by CMC, LMC and HCV strategies (mashups managed by the cloud).

tual Things, reducing the number of hops between workspaces, were found. LMC and HCV strategies were not able to reduce c^2 because these strategies are conditioned in their search for the cheapest materialization, as just mentioned.

Regarding the impact of α, it is possible to conclude that strategies present more similar costs when c^1 has more importance than c^2 ($\alpha = 0.75$) because of the just mentioned improvement of c^2 by CMC. Note that HCV presents



(a) c^2 using $\alpha = 0.25$



(b) c^2 using $\alpha = 0.75$.

Figure 5.7: C2 component of materialization cost obtained by CMC, LMC and HCV strategies (mashups managed by the cloud).

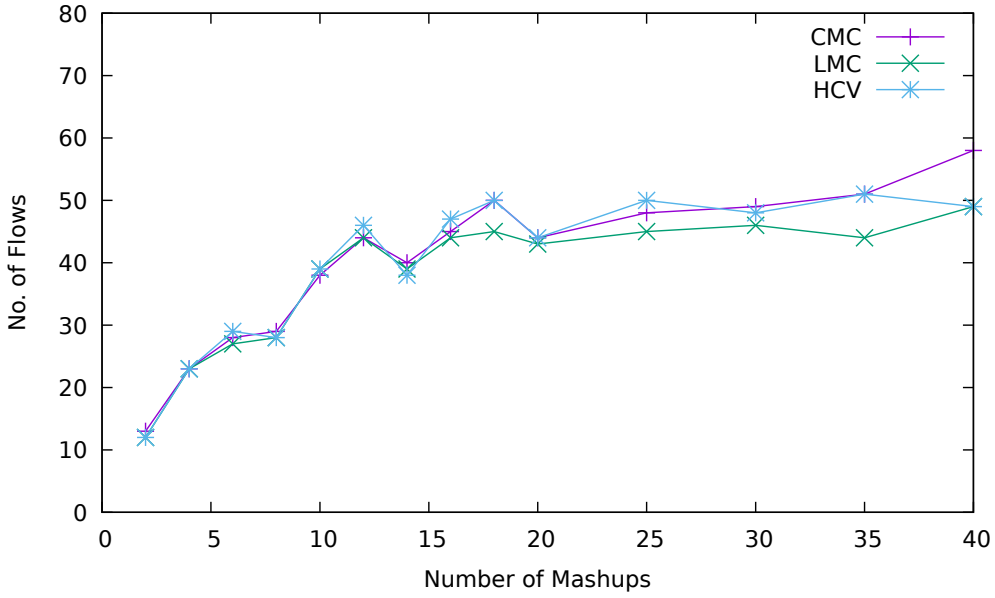
the best results on c^1, when compared with the other strategies, being able to make more aggregations without increasing the materialization cost. This strategy may, however, be improved in the future for better virtual Thing placements to be found, as its c^2 does not reduce as in CMC.

Number of Flows

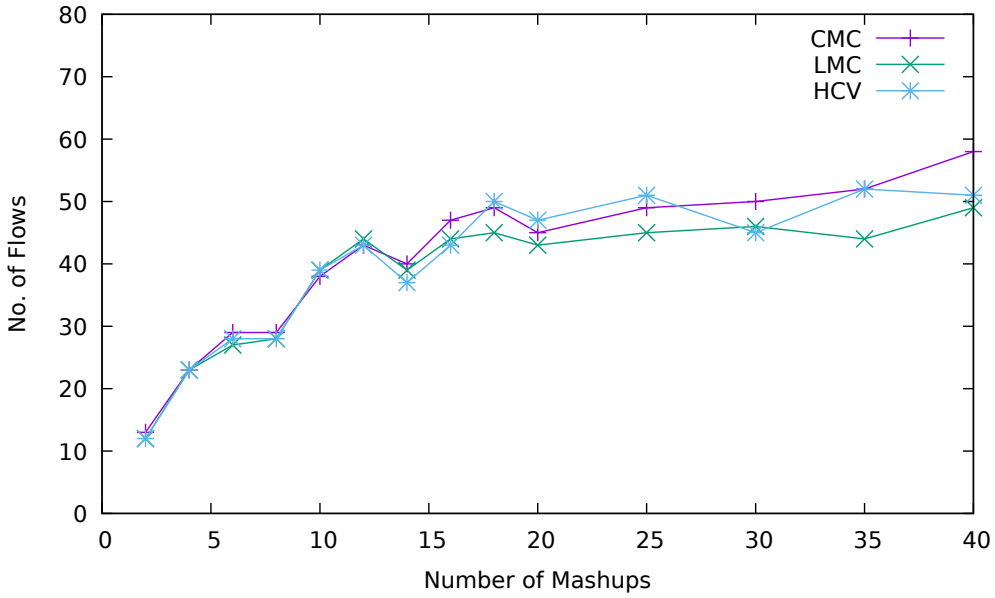
From the number of flows between virtual Thing workspaces, plotted in Figure 5.8, it is possible to conclude that the strategies with higher aggregation level are able to bind additional mashup elements to virtual Things, after all the devices are in use, without much impact on the number of flows. Although more mashup elements are being fulfilled, the aggregations do not increase the number of flows because virtual Things are the ones exchanging flows, and not individual mashup elements.

5.2.6 Conclusions

A resource allocation model for Se-aaS business models was here addressed. The model fits multiple emerging IoT Se-aaS business models, including the ones where client applications have bindings to mashups in the cloud, each mashup combining one or more devices. This way, applications can share devices registered in the cloud, for their mashups to operate, using cloud and device resources more effectively. The advantage of managing mashups in the cloud, instead of managing them at the client side, is that delays associated with multiple travelling sessions of data to the client are avoided. A heuristic was also proposed, having the resource allocation model as a basis that allows for the implementation of strategies leading to an efficient allocation of resources. The strategy with the best performance is HCV because devices are used for the materialization of virtual Things with more mashup elements mapped to it, while fulfilling more mashup elements. HCV picks the cheapest materialization from mashup elements with highest cost variance, based on the assumption that their late selection could significantly increase the overall cost. However, mashup elements having high cost variance end up being the ones with more compatible requirements, leading to a higher level of aggregation.



(a) $\alpha = 0.25$



(b) $\alpha = 0.75$

Figure 5.8: Number of flows obtained by CMC, LMC and HCV strategies (mashups managed by the cloud).

5.3 Mathematical Programming Formalization

5.3.1 Definitions and Assumptions

A CSP, denoted by $\mathcal{S}$, includes a set of distributed networks that interconnect to provide services, and these can be organized according to a common role or in order to better serve certain regions. Therefore, $\mathcal{S} = \{\mathcal{S}_1, ...\mathcal{S}_{|\mathcal{S}|}\}$, where $\mathcal{S}_i$ includes a set of computing resources. The set of all applications, requesting

for registered physical Things, is denoted by $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_{|\mathcal{A}|}\}$, and these are assumed to be outside the cloud. An application $\mathcal{A}_i$ can have one or more independent components, denoted by $\mathcal{C}(\mathcal{A}_i) = \{\mathcal{C}_1^i, ..., \mathcal{C}_{|\mathcal{C}(\mathcal{A}_i)|}^i\}$, and each component $\mathcal{C}_j^i$ is binded to a mashup in the cloud.

Definition 9 (Mashup)**.** *Workflow wiring together a set of elements/nodes. A mashup element n has a functionality requirement and a set of property conditions, denoted by $\bar{f}_n$ and $\bar{\mathcal{P}}_n$, respectively. The output of a mashup element can be input to another, while final mashup output data is sent to the client.*

It is assumed that clients use specific templates, at the CSP, to draw the mashup associated with each application component. Thus, mashup elements are not physical Things but, instead, elements that specify requirements/conditions using vocabularies (see [10]). That is, functionality requirement and property conditions are semantic-based. Each $p_n \in \bar{\mathcal{P}}_n$ has a "subject-predicate-object" description[1] of the condition/requirement that is being defined (e.g., cameraResolution greaterThan 12.1MP; frequencySampling equalTo 10s). The overall population of mashup elements (from all applications) is denoted by $\mathcal{N}$.

A set of physical Things, denoted by $\mathcal{T}^{\mathrm{P}}$, is assumed to be registered at the cloud. The owners voluntarily register/deregister physical Things to/from the cloud, meaning that CSPs must compensate the device owners for their contribution, or find some incentive mechanism for them to participate [44, 45].

Definition 10 (Physical Thing)**.** *A sensor detecting events/changes, or an actuator receiving commands for the control of a mechanism. The model of a physical Thing i includes all properties necessary to describe it, denoted by $\mathcal{P}_i$, and a functionality, denoted by f_i. That is, $\mathcal{P}_i = \{p : p \in \mathcal{P}\}$ and $f_i \in \mathcal{F}$, where $\mathcal{P}$ is the overall set of properties (e.g., sensing range, communication facility, energy consumption, location), and $\mathcal{F}$ the overall set of functionalities (e.g., image sensor).*

Each property $p_i \in \mathcal{P}_i$ has a "subject-predicate-object" description associated with it (e.g., cameraResolution hasValue 12.1MP).

[1] A Resource Description Framework (RDF) triple. See [1].

Instead of directly matching mashup elements to devices, virtual Things are used. More specifically, each mashup element $n \in \mathcal{N}$ will be mapped to a single virtual Thing, while a virtual Thing can be mapped to multiple mashup elements (with same functionality and compatible property requirements), and virtual Things are the ones to be "materialized" onto physical Things.

Definition 11 (Virtual Thing)**.** *Thing built at the cloud that acts on behalf of a set of mashup elements. The materialization of a virtual Thing j must fulfil the requirements of all its mashup elements.*

The use of virtual Things, each requiring some virtual workspace, allows data to be consumed by multiple application mashup elements while reducing data collection/storage and increasing the usefulness of data. In other words, resources can be better utilized. Different matchings of mashup elements to virtual Things will have different impacts on resource usage, cloud scalability and Quality of Experience (QoE). The set of virtual Things created inside the cloud is denoted by $\mathcal{T}^{\mathrm{V}}$.

The use of semantic tools allows the cloud to find different ways of achieving a functionality. That is, a functionality $f \in \mathcal{F}$ can be achieved by joining functionalities at multiple devices. Thus, there will be multiple ways of achieving a functionality, and each one of them can be materialized in one or more devices. The set of possible materializations for functionality f is denoted by $\mathcal{M}(f)$, and $\mathcal{M}_i^f \in \mathcal{M}(f)$ denotes the i^{th} possible materialization, which may include one or more devices. That is[2], $\cup_{t \in \mathcal{M}_i^f} f_t \triangleq f$, $\forall \mathcal{M}_i^f \in \mathcal{M}(f)$. A virtual Thing will be materialized using one of these materialization possibilities. The Se-aaS materialization problem is defined as follows.

Definition 12 (Se-aaS Materialization (SSM) Problem)**.** *Given a set of applications, each with a set of components binded to mashups, assign mashup elements to virtual Things, and materialize virtual Things onto physical Things, so that the overall cost is minimized while meeting the functionality and property needs of mashup elements.*

Regarding the just mentioned cost, let us assume that $\eta = \{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_{|\mathcal{T}^{\mathrm{V}}|}\}$ is a feasible partition of mashup elements (all elements in a $\mathcal{N}_j$ have the same

[2]The symbol $\triangleq$ means equal by definition, in our case logically/semantically equivalent.

functionality requirement and compatible property requirements). That is, each $\mathcal{N}_j$ will give rise to a virtual Thing. The cost of assigning materialization $\mathcal{M}_i^f$ to $\mathcal{N}_j$, $\mathcal{M}_i^f \in \mathcal{M}(f)$, will be

$$Cost(\mathcal{M}_i^f, \mathcal{N}_j) = \sum_{\{p \in \chi\}} min_{n \in \mathcal{N}_j}\{\Delta_{f,i}^{n,p}\}, \tag{5.7}$$

where $\chi = \cup_{n' \in \mathcal{N}_j} \bar{\mathcal{P}}_{n'}$ and $\Delta_{f,i}^{n,p}$ is the highest gap between the property requirement p at mashup element n, and the value of property p offered by every physical Thing in $\mathcal{M}_i^f$. This is so because more than one physical Thing in $\mathcal{M}_i^f$ can have a given property. Then, for each property $p \in \chi$ the *min* is used to capture the lowest gap between requirements and physical Thing's tolerance/acceptance for the property in question.

5.3.2 Problem Formalization

The following client related information is assumed to be known:

$\mathcal{A}$	Set of applications, where $\mathcal{A}_i \in \mathcal{A}$ refers to a specific application.
$\mathcal{C}(\mathcal{A}_i)$	Set of independent application components at $\mathcal{A}_i \in \mathcal{A}$, where $\mathcal{C}_j^i \in \mathcal{C}(\mathcal{A}_i)$ refers to a specific component. A component is binded to a mashup in the cloud, each mashup having a set of nodes/elements.
$\mathcal{N}$	Set of all mashup elements, from all application components. That is, $\mathcal{N} = \cup_{\mathcal{A}_i \in \mathcal{A}} \mathcal{C}(\mathcal{A}_i)$.
$\bar{f}_n$	Functionality required by mashup element $n \in \mathcal{N}$, $\bar{f}_n \in \mathcal{F}$.
$\bar{\mathcal{P}}_n$	Set of all property conditions of mashup element $n \in \mathcal{N}$.
$\Phi^{n,n'}$	One if $n, n' \in \mathcal{N}$ have different functionalities or some property that makes then incompatible for materialization onto the same physical Thing; zero otherwise.
$\Omega^{n,n'}$	One if there is a mashup flow $n \rightarrow n'$, $n, n' \in \mathcal{N}$; zero otherwise.

That is, each $n \in \mathcal{N}$ requires a functionality and imposes several property constraints, which imposes limitations on the set of mashup elements to be assigned to a virtual Thing and, consequently, its materialization onto a physical Thing. Some n's may have no functionality and property requirements if used for aggregation of flows with web services. Regarding physical Things, the known related information is the following:

$\mathcal{T}^{\mathrm{P}}$	Set of all physical Things registered at the cloud, where $t \in \mathcal{T}^{\mathrm{P}}$ is used to refer to a specific physical Thing.
$\mathcal{P}_t$	Set of properties of physical Thing $t \in \mathcal{T}^{\mathrm{P}}$.
f_t	Functionality of physical Thing $t \in \mathcal{T}^{\mathrm{P}}$.
$\mathcal{M}(f)$	Set of possible materializations for functionality f, where $\mathcal{M}_i^f \in \mathcal{M}(f)$ denotes the i^{th} possible materialization, which may include one or more devices.
$\Delta_{f,i}^{n,p}$	Highest gap value, from all physical Things enrolled in materialization $\mathcal{M}_i^f$, for a particular p of $n \in \mathcal{N}$.
$\Delta^{\max}$	Highest possible property gap.

Note that $\Phi^{n,n'}$ and $\Delta_{f,i}^{n,p}$ can be extracted using SPARQL because both property/functionality requirements and physical Thing acceptance are semantic-based.

The variables required to formulate the SSM problem will be:

α_i^f	One if the i^{th} possible materialization for functionality f, $\mathcal{M}_i^f \in \mathcal{M}(f)$, is being used; zero otherwise. A virtual Thing is allocated to $\mathcal{M}_i^f$, and active, if materialization $\mathcal{M}_i^f$ is being used.
$\kappa_{f,i}^t$	One if physical Thing $t \in \mathcal{T}^{\mathrm{P}}$ is enrolled in the materialization of virtual Thing $\mathcal{M}_i^f$; zero otherwise.
$\beta_{f,i}^n$	One if mashup element $n \in \mathcal{N}$ is mapped to virtual Thing $\mathcal{M}_i^f$; zero otherwise.
$\zeta_{f,i}^p$	Highest gap associated with property p, from all $n \in \mathcal{N}$, at virtual Thing $\mathcal{M}_i^f$.
$\rho_{f',i'}^{f,i}$	One if there is flow from virtual Thing $\mathcal{M}_i^f$ to virtual Thing $\mathcal{M}_{i'}^{f'}$, zero otherwise.
Υ	Total gap cost.
Ψ	Total number of flows.

The SSM problem is mathematically formulated as follows.

– Objective function:

$$Maximize \sum_{\{f \in \mathcal{F}\}} \sum_{\{\mathcal{M}_i^f \in \mathcal{M}(f)\}} \sum_{\{n \in \mathcal{N}\}} \beta_{f,i}^n - \frac{\Upsilon}{P \times \Delta^{\max} \times N} - \\ - \frac{\Psi}{P \times \Delta^{\max} \times N^3} \tag{5.8}$$

where $P = \sum_{n \in \mathcal{N}} |\bar{\mathcal{P}}_n|$, $N = |\mathcal{N}|$ and $\Delta^{\max}$ is the highest possible property gap. With this goal the number of fulfilled mashup elements is first maximized and then, as a secondary goal, the total gap cost and total number of flows (between virtual Things inside the cloud) is minimized, as these have are components with negative sign. This goal is subject to:

– Physical Thing assignment:

$$\kappa_{f,i}^{t} \geq \alpha_{i}^{f}, \forall f \in \mathcal{F}, \forall \mathcal{M}_{i}^{f} \in \mathcal{M}(f), \forall t \in \mathcal{M}_{i}^{f} \tag{5.9}$$

$$\sum_{\{f \in \mathcal{F}\}} \sum_{\{\mathcal{M}_{i}^{f} \in \mathcal{M}(f)\}} \kappa_{f,i}^{t} \leq 1, \forall t \in \mathcal{T}^{\mathrm{P}} \tag{5.10}$$

Constraints (5.9) state that all physical Things enrolled in a materialization (of a virtual Thing) must be in use if the virtual Thing is active. Constraints (5.10) force a physical Thing not to be assigned to more than one virtual Thing.

$$\sum_{\{f \in \mathcal{F}\}} \sum_{\{\mathcal{M}_{i}^{f} \in \mathcal{M}(f)\}} \sum_{\{t \notin \mathcal{M}_{i}^{f}\}} \kappa_{f,i}^{t} \leq 0 \tag{5.11}$$

Constraints (5.11) state that a physical Thing can not be assigned to a virtual Thing if it is not participating in the materialization (device orchestration) of such virtual Thing.

– Ensuring functionality of mashup elements:

$$\sum_{\{\mathcal{M}_{i}^{f} \in \mathcal{M}(f)\}} \beta_{f,i}^{n} \leq 1, \forall n \in \mathcal{N}, f = \bar{f}_{n} \tag{5.12}$$

$$\alpha_{i}^{f} \geq \beta_{f,i}^{n}, \forall n \in \mathcal{N}, f = \bar{f}_{n}, \forall \mathcal{M}_{i}^{f} \in \mathcal{M}(f) \tag{5.13}$$

Constraints (5.12) ensure that the functionality required by n is fulfilled by no more than one virtual Thing. In constraints (5.13), virtual Things become active if at least one mashup element is mapped to it.

– Ensuring property conditions of mashup elements:

$$\beta_{f,i}^{n} + \beta_{f,i}^{n'} \leq 2 - \Phi^{n,n'}, \forall n, n' \in \mathcal{N}, f = \bar{f}_{n}, \\ , \forall \mathcal{M}_{i}^{f} \in \mathcal{M}(f), \tag{5.14}$$

These are used to ensure that no two mashup elements with incompatible properties are materialized onto the same physical Thing.

– Mashup element requirements to physical property gaps:

$$\zeta_{f,i}^{p} \geq \beta_{f,i}^{n} \times \Delta_{f,i}^{n,p} - (1 - \beta_{f,i}^{n}) \times \Delta^{\max}, \forall n \in \mathcal{N}, f = \bar{f}_n, \\ , \forall \mathcal{M}_i^f \in \mathcal{M}(f), \forall p \in \bar{\mathcal{P}}_n \tag{5.15}$$

where $\Delta_{f,i}^{n,p}$ is the highest gap value, from all physical Things in a $\mathcal{M}_i^f$, for a particular p of n (more than one physical Thing used in $\mathcal{M}_i^f$ can have property p), and $\Delta^{\max}$ is the highest possible gap value. Thus, for a set of mashup elements mapped to a virtual Thing, to be materialized, $\zeta_{f,i}^{p}$ will be an upper bound for such highest gap values.

$$\Upsilon = \sum_{\{f \in \mathcal{F}\}} \sum_{\{\mathcal{M}_i^f \in \mathcal{M}(f)\}} \sum_{\{p \in P\}} \zeta_{f,i}^{p} \tag{5.16}$$

where Υ is included in the objective function, for gap cost minimization.

– Flows between virtual things:

$$\rho_{f',i'}^{f,i} \geq (\beta_{f,i}^{n} + \beta_{f',i'}^{n'}) \times \Omega^{n,n'} - 1, \forall n, n' \in \mathcal{N}, \\ , f = \bar{f}_n, f' = \bar{f}_{n'}, \forall \mathcal{M}_i^f \in \mathcal{M}(f), \forall \mathcal{M}_{i'}^{f'} \in \mathcal{M}(f') \tag{5.17}$$

where $\Omega^{n,n'}$ is given information stating if there is a mashup flow $n \rightarrow n'$. These constraints find if there is any flow between any two virtual Things, which depends on the mashup element to virtual Thing mapping. The overall number of flows between virtual Things is given by:

$$\Psi = \sum_{\{f \in \mathcal{F}\}} \sum_{\{\mathcal{M}_i^f \in \mathcal{M}(f)\}} \sum_{\{f' \in \mathcal{F}\}} \sum_{\{\mathcal{M}_{i'}^{f'} \in \mathcal{M}(f')\}} \rho_{f',i'}^{f,i} \tag{5.18}$$

which is included in the objetive function, for minimization.

– Non-negativity assignment to variables:

$$\alpha_i^f, \kappa_{f,i}^t, \beta_{f,i}^n, \rho_{f',i'}^{f,i} \in \{0,1\}; \zeta_{f,i}^p, \Upsilon, \Psi \in \Re^+. \tag{5.19}$$

The CPLEX[3] optimizer is used to solve instances of this problem. The solution found will be the optimal solution for the SSM problem instance under consideration.

5.3.3 Hardness of the Problem

Theorem 1. *The SSM problem is NP-hard.*

Proof. Considering a compatibility graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ such that $(n, n') \in \mathcal{E}$ iff $\Phi^{n,n'} = 0$, constraints (5.14) state that all mashup elements assigned to a virtual Thing (to be materialized onto a device) must be compatible, which corresponds to a clique subgraph in $\mathcal{G}$. When adding constraints (5.15)-(5.18), and inserting Υ and Ψ into the objective function, the clique size is to be maximized as this leads to lower Υ and Ψ values. This comes down to the maximum clique problem, which happens to be NP-hard [55]. Since multiple virtual Things are being built, the SSM problem can be seen as a multi-dimensional maximum clique problem. The SSM problem is, therefore, NP-hard.

[3]IBM ILOG CPLEX Optimizer.

5.4 Algorithmic Approach

5.4.1 Motivation

Although optimal solutions can be obtained using the mathematical model previously presented, the hardness of the SSM problem makes it difficult to obtain solutions for larger instances of the problem. This is why a heuristic is proposed in [21]. Such research work is relevant because, to the best of our knowledge, it is the first addressing mashups managed in the cloud. However, the heuristic proposed in [21] has the following drawbacks:

- Maximum cliques are extracted from the compatibility graph (see Theorem 1), one for each mashup element $n \in \mathcal{N}$. However, picking maximum cliques is only feasible through heuristic techniques [55]. Among existing heuristic techniques, the greedy-based ones present great simplicity and high speed but have difficulty in finding good solutions given its myopic nature.
- Only virtual Things (and corresponding materialization) for maximum cliques are considered. The approach is to process all costs for all possible maximum clique materializations first, and then some criteria is used to select the cheapest materializations. However, the impact of selecting a certain materialization, at each step, is not evaluated. That is, selecting a certain materialization first can make one or more future materializations infeasible because they compete for devices. Also, no other mashup element groupings (subgraphs of maximum cliques) are explored, meaning that mashup elements may not be fulfilled although other mashup element groupings could be viable.

Due to the just mentioned drawbacks, a heuristic is proposed here that tries to increase the size of cliques around each mashup element, in parallel, while performing productive swap operations. This way different mashup element groupings are evaluated while avoiding building a myopic greedy approach.

5.4.2 Algorithm Details

Let us assume the previously mentioned compatibility graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ such that $\{n, n'\} \in \mathcal{E}$ iff $\Phi^{n,n'} = 0$, and let $\mathcal{N}_n$ denote the neighbours of node $n \in \mathcal{N}$, $\mathcal{N}_n = \{n' \in \mathcal{N} : (n, n') \in \mathcal{E}\}$. Let us define $\mathcal{X}(\mathcal{C})$ as the set of nodes that may expand a given clique either through direct inclusion or after a swap operation.

That is, considering a given clique $\mathcal{C} \subset \mathcal{G}$, $\mathcal{X}(\mathcal{C}) = \{n \in \mathcal{N} : |\mathcal{C} \backslash \mathcal{N}_n| \in \{0, 1\}\}$. When $|\mathcal{C} \backslash \mathcal{N}_n| = 0$ the clique can expand by the direct inclusion of node n, and when $|\mathcal{C} \backslash \mathcal{N}_n| = 1$ the clique can swap one of its nodes (the one not connected to n) with n in order to diversify the attempts to expand.

As previously stated, the heuristic tries to increase the size of cliques around each mashup element, in parallel. Thus, every node will be included in a clique, although with a single node at the beginning. When attempting to expand cliques, nodes may move from one clique to another and a clique can be absorbed by another.

Claim 1. *The direct inclusion of $n \in \mathcal{X}(\mathcal{C}^1)$ into clique $\mathcal{C}^1$, and consequent removal from its current clique $\mathcal{C}^2$, should only be performed if $\exists \mathcal{M}_i^f, \mathcal{M}_j^f \in \mathcal{M}(f)$ such that $Cost(\mathcal{M}_i^f, \mathcal{C}^1 \cup \{n\}) + Cost(\mathcal{M}_j^f, \mathcal{C}^2 \backslash \{n\}) < Cost(\mathcal{M}_i^f, \mathcal{C}^1) + Cost(\mathcal{M}_j^f, \mathcal{C}^2)\}$, where f is the functionality required by all nodes in $\mathcal{C}^1$ and $\mathcal{C}^2$. That is, there is an overall cost reduction.*

Claim 2. *The swap between $n \in \mathcal{X}(\mathcal{C}^1)$ and $n' \in \mathcal{C}^1$, where $(n, n') \notin \mathcal{E}$, should only be performed if $\exists \mathcal{M}_i^f, \mathcal{M}_j^f \in \mathcal{M}(f)$ such that applying Claim 1 to $\mathcal{C}^1 \backslash \{n'\} \cup \{n\}$ compensate the cost increase associated with such swap.*

That is, the idea in Claim 2 is that there will be a cost increase when swapping n with n' (current cliques were built having the best cost into account, and at the time all neighbours were evaluated), but then a direct inclusion compensates it.

5.4.3 Performance Analysis

Scenario Setup

For the evaluation of results, random scenarios were generated based on a pool of functionalities and a pool of properties, for each functionality. The physical Things and mashup elements were created as follows:

- Mashups are randomly generated using the algorithm in [39], considering an average number of elements per mashup.
- A mashup element will have its functionality requirement randomly selected from the pool of functionalities, together with a percentage of its properties.

Algorithm 3: Clique expansion based resource allocation heuristic (mashups managed by cloud).

```
1  /* STEP: Initialization */
2  C = {C^1, C^2, ..., C^|N|}
3  i = 1
4  for each n ∈ N do
5  |  C^i ← n; i++.
6  end
7  /* STEP: Clique expansion */
8  repeat
9  |  L = ∅
10 |  for each C^i ∈ C : C^i ≠ ∅ do
11 |  |  Determine X(C^i)
12 |  |  for each n ∈ X(C^i) do
13 |  |  |  if |C^i \ N_n| = 0 then
14 |  |  |  |  L ← cost redution scenario using Claim 1
15 |  |  |  end
16 |  |  |  if |C^i \ N_n| = 1 then
17 |  |  |  |  L ← cost redution scenario using Claim 2
18 |  |  |  end
19 |  |  end
20 |  end
21 |  l* = BestScenario(L)
22 |  Realize l*
23 until l* = ∅;
```

- Any pair $n_i, n_j \in \mathcal{N}$ with the same functionality requirement, at at least with a property in common, is compatible with probability δ (incompatibility may exist due to their property conditions).
- A physical Thing has a functionality assigned to it, randomly selected from the pool of functionalities, together with all the properties associated with the selected functionality.
- The gap between a property condition and device property is randomly selected from $\{\Delta_1, ..., \Delta_5\}$, where Δ_1 is the lowest cost and Δ_5 is the highest.

Table 5.2 summarizes the adopted parameter values.

Note that, if mashups were outside the cloud, then more flows, between application components and the cloud in this case, would exist. The number of flows would be $No.\ of\ fulfilled\ mashup\ elements \times 2$ (see Figure 5.9). These values are not shown in order not to disturb the visualization of results from strategies. Flows would also have higher transfer delays, as these are not confined to internal transfers inside the cloud.

Table 5.2: Simulation setup to evaluate the mathematical programming model and heuristics (mashups managed by the cloud).

Parameter	Value
Functionality pool size	10
Avg size of property pools	10
Total number of devices	100
Avg number of elements per mashup	10
Mashup element's properties (from pool)	50%
δ	0.5
$\{\Delta_1, ..., \Delta_5\}$	$\{1, ..., 5\}$

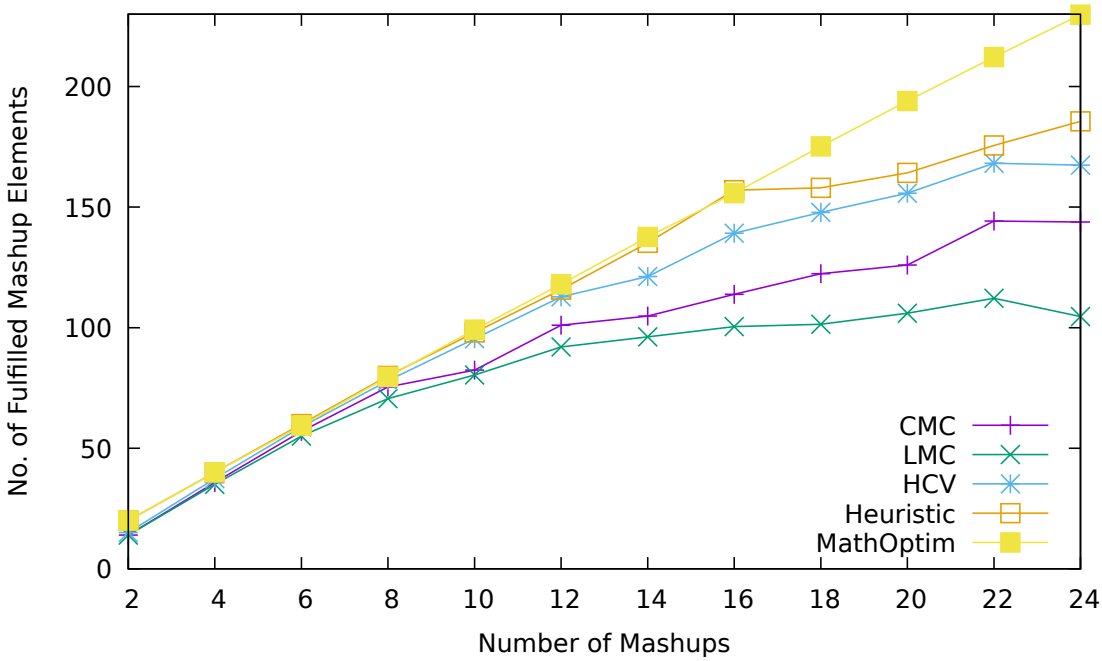


Figure 5.9: Mathematical programming model and heuristics: Number of fulfilled mashup elements (mashups managed by the cloud).

Fulfilled Mashup Elements and Virtual Things

Here, the number of mashup elements that have been fulfilled (mapped to a virtual Thing), the total number of generated virtual Things (successfully materialized onto devices), and the average number of mashup elements per virtual Thing (average size of cliques in compatibility sub-graph $\mathcal{G}$), are analysed. These results are shown in plots of Figures 5.9, 5.10 and 5.11, respectively, for an increasing number of mashups. Regarding the mathematical model, and after CPLEX obtain the solution for instances, the values for the first plot are the result of counting non-zero β variables, the values for the second plot are the result of counting non-zero α variables, while for the last plot the values are the result of dividing the first counting by the second.

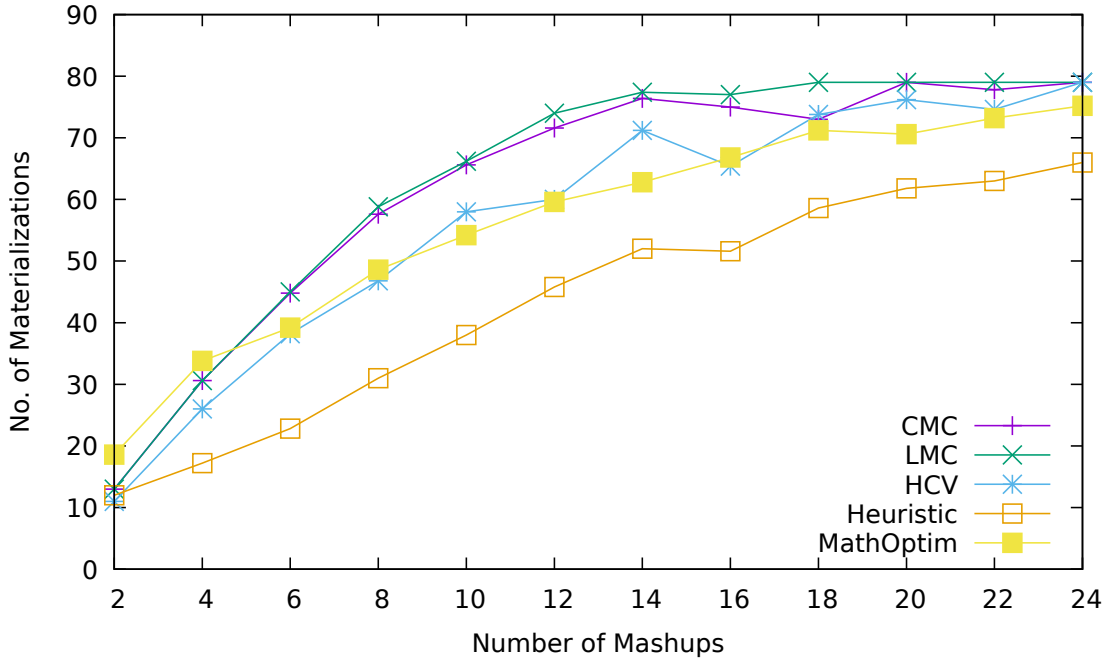


Figure 5.10: Mathematical programming model and heuristics: Number of virtual Things materialized onto physical devices (mashups managed by the cloud).

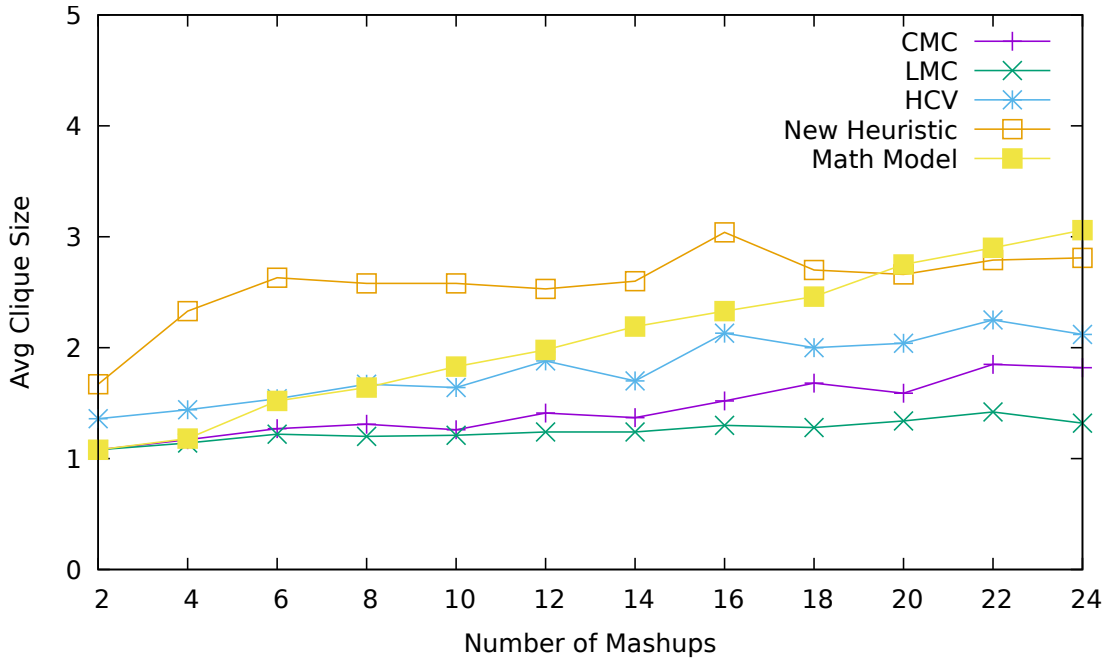


Figure 5.11: Mathematical programming model and heuristics: Average number of mashup elements per virtual Thing (mashups managed by the cloud).

From these plots it is possible to observe that the new heuristic is able to fulfill more mashup elements (see plot 5.9) than all variants of the heuristic approach proposed in [21], closely approaching the mathematical optimization model. This is more pronounced when there are less than 16 mashups,

which is when the number of devices under utilization (materializations) is below 60% of the total number of available devices (see plot 5.10). When the number of materializations (virtual Things) gets closer to the number of available devices, the proposed heuristic slows down its performance, although it still outperforms the heuristic variants from [21]. Please note that the proposed heuristic could increase the number of fulfilled elements, for such scenarios, if swap operations explore more distant neighbourhoods. However, this will increase the complexity of the heuristic.

Regarding the number of virtual Things (materializations), the proposed heuristic outperforms all approaches, mathematical model included. The heuristic is able to use less virtual Things (materializations), meaning that more mashup elements are mapped to a virtual Thing (cliques of larger size), making materializations more effective and releasing more devices for future materializations (see plot 5.11). Note that the heuristic can have better results than the mathematical optimization model because the primary goal of the mathematical model is to maximize the number of fulfilled mashup elements, and not the clique size. To maximize the clique size the mathematical model would become non-linear, making the model untreatable. However, since the materialization cost is also at the objective function (secondary component), the mathematical model has also interest in larger clique sizes. This becomes noticeable when the population of mashup elements to be fulfilled increases. In this case multiple solutions exist for the same number of fulfilled mashup elements, and then the second component can play its role. For this reason, the mathematical model keeps increasing the clique size as the number of mashups increases, being able to manage materializations more efficiently than the other approaches in such scenarios: high number of fulfilled mashup elements and large clique size.

Materialization Cost

Here, the total cost associated with the chosen materializations, calculated using Eq. 5.7, and average cost per mashup element, are analysed. These results are shown in plots of Figures 5.12 and 5.13, respectively, for an increasing number of mashups. Regarding the mathematical model, and after CPLEX obtain the solution for instances, the value of Υ is used at the first plot, while at the last plot the value is the result of dividing the Υ value by the number of fulfilled mashup elements, from Figure 5.9.

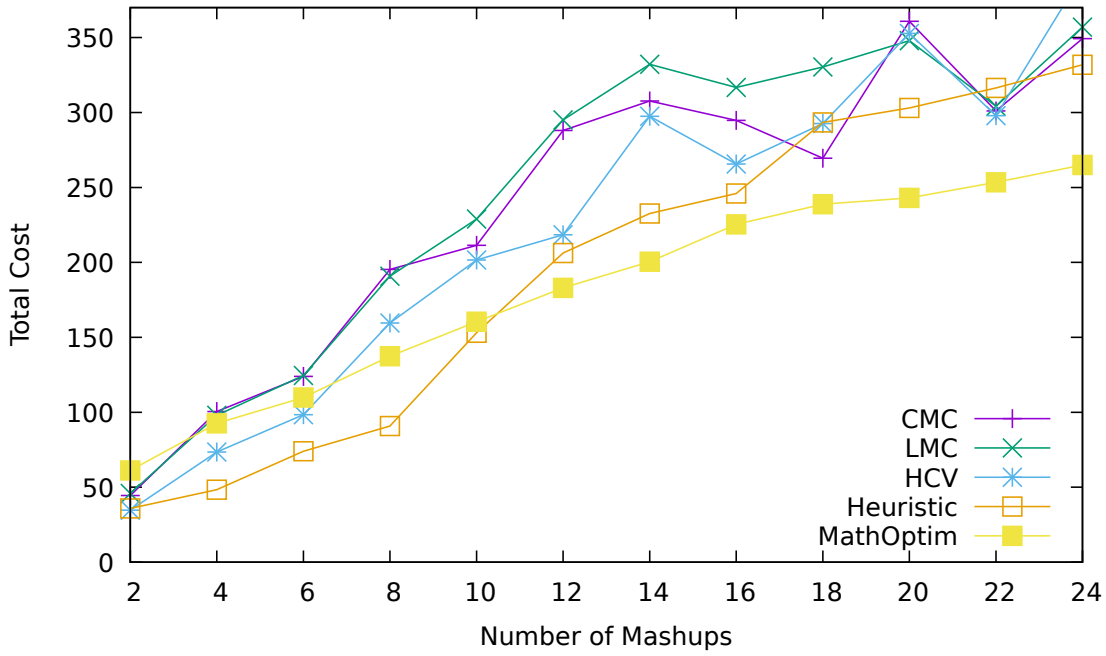


Figure 5.12: Mathematical programming model and heuristics: Total cost of materializations (mashups managed by the cloud).

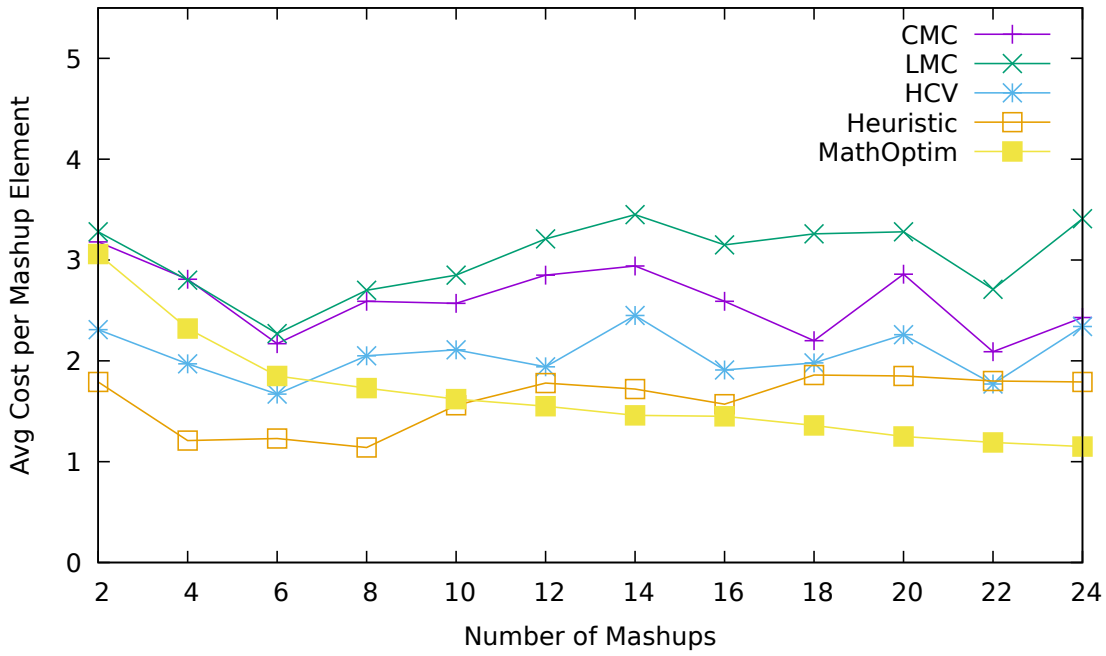


Figure 5.13: Mathematical programming model and heuristics: Average cost per mashup element (mashups managed by the cloud).

From the results it is possible to observe that the mathematical optimization model is the one finding devices more close to the needs of mashup elements, resulting into lower materialization costs, releasing the other devices for future demands. This is so because the cost is included in the objective

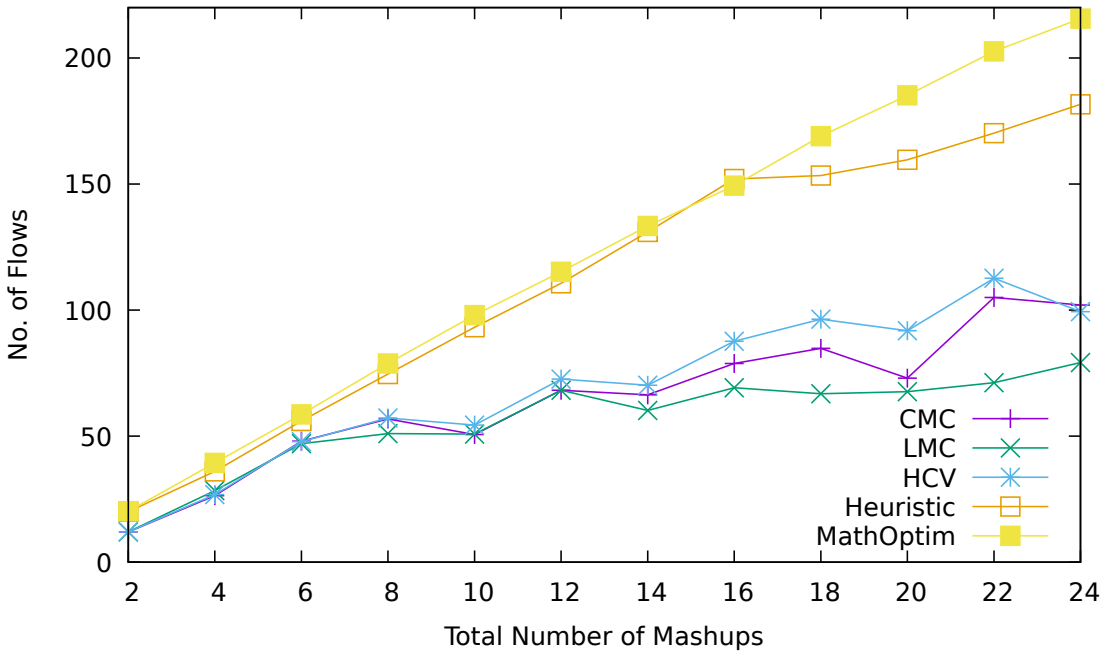


Figure 5.14: Mathematical programming model and heuristics: Total number of flows (mashups managed by the cloud).

function, although as a secondary goal. Its effect becomes more clear when the population of mashup elements increases. That is, since more solutions exist for the same value of the first component (number of fulfilled mashup elements), the optimizer then retrieves the solution having lower cost.

The proposed heuristic presents better results (lower materialization costs) than the variants of the approach proposed in [21], meaning that materializations are choosing devices closer to the needs of the mashup elements.

Flows at the Cloud

Here, the total number of flows between virtual Things stored at the cloud, and average number of flows per mashup element, are analysed. These results are shown in plots of Figures 5.14 and 5.15, respectively, for an increasing number of mashups. Regarding the mathematical model, and after CPLEX obtain the solution for the instances, the value of Ψ is used in the first plot, while at the last plot the value is the result of dividing Ψ by the number of fulfilled mashup elements, from Figure 5.9.

Results show that the proposed mathematical optimization model and heuristic do not reduce flows as the number of fulfilled mashup elements per virtual Thing (materialization) increases. This means that cliques are built

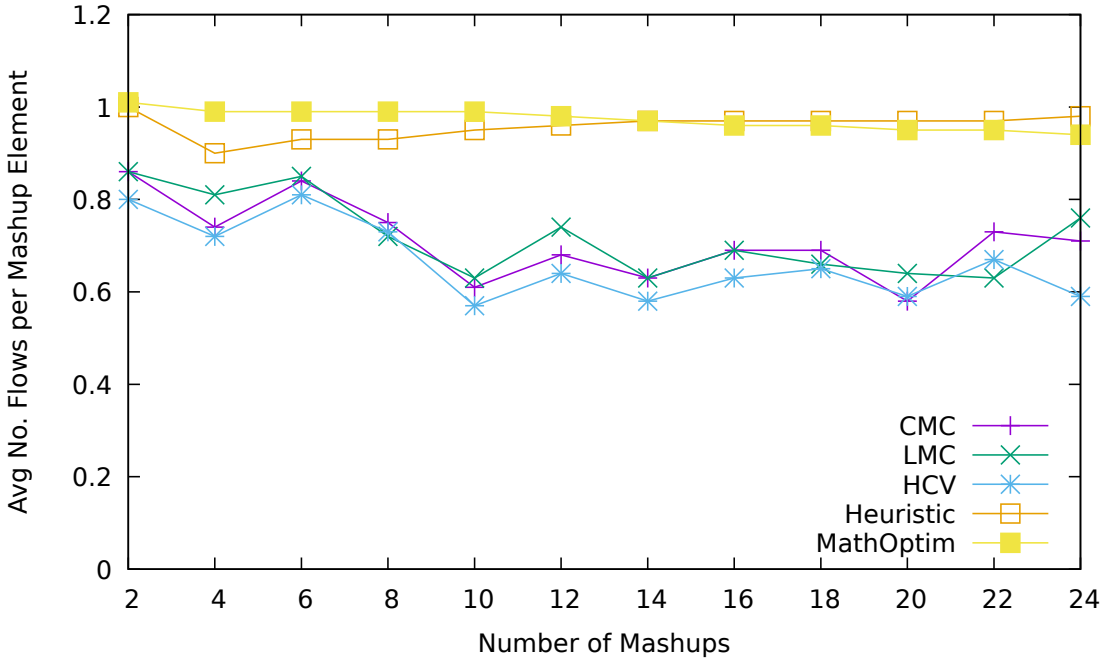


Figure 5.15: Mathematical programming model and heuristics: Average number of flows per mashup element (mashups managed by the cloud).

mainly having as goal the increase of fulfilled mashup elements, and taking into account the materialization cost, and no attempt exists to build cliques with mashup elements having common predecessor/successors. Therefore, the third component at the objective function of the mathematical optimization model ends up having no impact, and turns out to be neglected. Contrariety to the heuristic approach proposed in [21], the heuristic proposed here does take into account flow dependency, otherwise all predecessors/successors of mashup elements, and reallocation of mashup elements at cliques, would have to be evaluated by the heuristic, which significantly increases its complexity. Two kinds of connectivity (can be seen as layers) would have to be considered: compatibility graph and flow dependency among mashup elements. The heuristic can, however, be easily extended to incorporate this, if it brings benefits for a particular cloud architecture.

5.4.4 Conclusions

The development of an adequate mathematical optimization model for this problem and another heuristic is proposed to optimize the size of cliques around each mashup element producing different mashup elements groups building a myopic greedy approach. The heuristic outperforms the known state-of-art method, fulfilling more mashup elements (requests from clients) and using less physical Things for materialization of virtual Things. This

means that virtual Things are acting on behalf of more mashup elements, making them more productive. The heuristic also shows lower materialization costs, which means that allocated physical Things are near to what is requested by clients, leaving physical Things with better features for future requests. The heuristic can be easily extended to take into account flow dependencies (predecessor/successors of its mashup elements) when building virtual Things, although this will increase its complexity.

CHAPTER 6

Conclusions and Future Work

6.1 Conclusions

The IoT is now attracting the attention from both academia and industry, and this interest is expected to grow [16, 41]. In IoT, physical objects can be accessed and controlled using electronic devices that are able to communicate using networking interfaces. However, the research in IoT is primarily driven by technological advances and not by applications or user needs. On the other hand, research on smart cities, smart transportation, and others, address specific problems and needs. An effective bridge between these two relies on an efficient resource discovery, access and management, which can be provided by what is now called the Web of Things, or WoT.

A move towards the WoT will prevent IoT from becoming just a collection of Things, unable to be discovered for interaction with others. The idea of WoT is to reuse and leverage readily available and widely popular web protocols, standards and blueprints to make data and services offered by objects accessible to a larger pool of web developers [22]. Also, device mashups can be more easily created, combining services/data from one or multiple physical Things with services/data from virtual web resources (e.g., multiple sensor data sources can be combined with virtual web resources to decide for an actuation at some device).

All developments mentioned above bring a chance for new business models to be created. The number and type of devices connected to the IoT will be huge, allowing large amounts of data to be collected and analysed.

As more and more physical Things become available, managed and accessed in the IoT world, and mashups are built, more data with processing needs will emerge, meaning that new challenges arise in terms of storage and

processing. A move towards to sensing in cloud infrastructure is needed to use processing and storage capabilities.

The Sensing as-a-Service (Se-aaS) model, relying on cloud infrastructures, emerges from this reality. Se-aaS is a business model built on cloud infrastructures, where sensor owners can register and de-register their devices, providing services and sensor data to application users (clients) in real time, with all the security and capabilities that cloud computing environments provide. The Se-aaS model has IoT infrastructure as a basis, and provides virtual sensors that are binded to one or more physical sensors, for multiple user applications to access data generated from multiple physical sensors.

In order to achieve these goals, the Se-aaS architecture relies on an infrastructure capable of responding to application/client requests, allowing the provisioning and managing of on-demand virtual sensors. The processing, monitoring and managing of all the system is ensured, together with the storing/retrieval of all sensing data. The Se-aaS platform should include the following functionalities:

- **Virtualization** - Virtual sensors binded to one or more physical Things so that they can be linked to multiple consumers;
- **Dynamic provisioning** - A virtual workspace/virtual machine provisioned on-demand that allows the creation of virtual sensors for consumers/clients/applications to use, control and retrieve data;
- **Multi-tenancy** - The Se-aaS platform should allow the sharing of data and sensors by consumers, while ensuring scalability, security and QoE.

Applications using Se-aaS platforms will have software components with bindings to virtual Things at the cloud, which creates a multi-user environment assisting in the use of resource-constrained physical wireless sensors.

In this thesis, several still not sufficiently addressed research topics of Se-aaS models were investigated. Namely: i) the allocation of sensors to respond to multiple applications and mashups; ii) the efficient assignment of mashup element clusters to physical Things; iii) scalability, elasticity and QoE of models; iv) delay of flows between workspaces and devices. In order to address these issues, various steps were done. These steps end up ensuring models and algorithms that respond to the just mentioned challenges. The steps were the following:

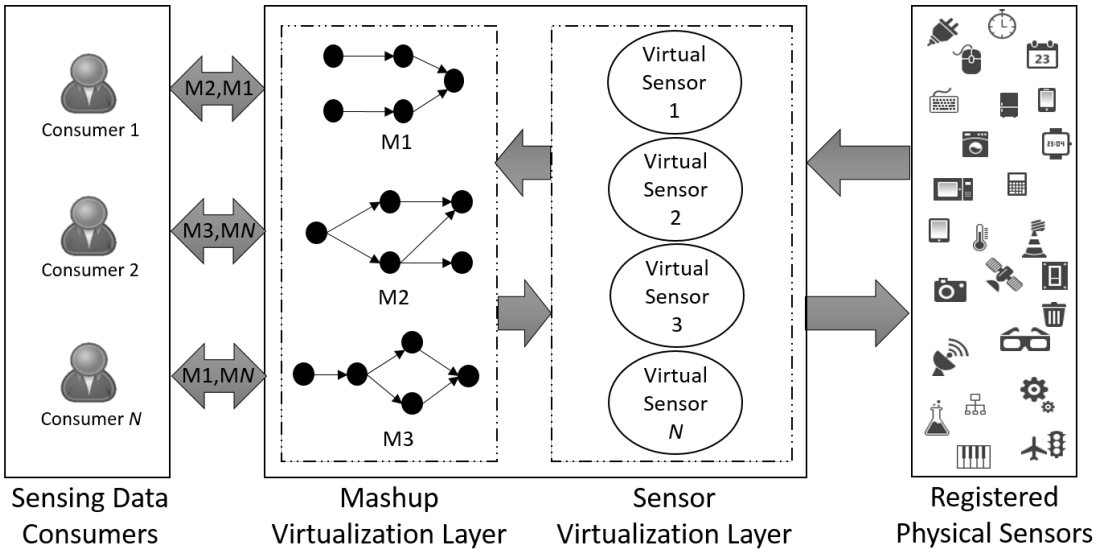


Figure 6.1: Se-aaS virtualization with mashups managed in the cloud.

- Development of a Se-aaS resource assignment model to evaluate the impact of resource allocation in scalability, elasticity and QoE;
- A mathematical formalization for the problem of selecting the best clusters os mashup elements (virtual Things) and their materialization, having costs as the basis of the decision;
- Mathematical models and heuristic algorithms to build the best clusters, considering two cases: mashups managed at the client and mashups managed at the cloud.
- A mathematical programming optimization model able to determine the optimal solution in resource allocation, which allowed the development of a new heuristic that outperforms the results obtained by the previous heuristic approaches.

The developments presented above are able to respond the open research issues in Se-aaS. In general, as shown in Figure 6.1, the first main objective was to improve the cloud's sensor virtualization layer by allocating the sensors with the adequate characteristics and properties to consumer application requests, while responding to multiple applications and mashups, leaving other sensors (with extra features) idle for future use. The second main objective was to incorporate in the model a mashup virtualization layer to decrease delays between cloud and data consumers, resulting in a faster and better experience to clients. The third and last main objective was to improve the overall resource allocation of the model by finding optimal solutions and

heuristics, for CSPs to be able to choose the adequate strategies for their specific cases, enhancing the services they provide.

In summary, all developments presented in this thesis had the goal of improving the resource allocation in the Se-aaS business model.

6.2 Future Work

The last developed heuristic algorithm, based on clique expansion, showed much lower materialization costs than the previously developed ones. This means that the allocated physical Things are nearer to client requests, leaving other physical Things (with extra features) idle for future use, which makes the model more resilient and able to fill more client requests.

As future work, this heuristic can be extended to take into account flow dependencies (predecessor/successors of its mashup elements) when building virtual Things, which will bring significant benefits not only to cloud architectures but also to users, which end up having a better quality of experience.

Bibliography

[1] W3C: ”Semantic Web W3C” - https://www.w3.org/standards/semanticweb/. Visualized in October 2019.

[2] *W3C: ”SPARQL Query Language for RDF”- https://www.w3.org/TR/rdf-sparql-query/*. Visualized in October 2019.

[3] web of things (WoT) architecture (editorial draft), 2018.

[4] AL-FAGIH, A. E., AL-TURJMAN, F. M., ALSALIH, W. M., AND HASSANEIN, H. S. Priced Public Sensing Framework for Heterogenous IoT Architectures. *IEEE Transactions on Emerging Topics in Computing 1*, 1 (2013), 133–147.

[5] AL-FUQAHA, A., GUIZANI, M., MOHAMMADI, M., ALEDHARI, M., AND AYYASH, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communication Surveys & Tutorials 17*, 4 (2015), 2347–2376.

[6] ANKUR, D., SOMA, K., AND K.P.CHETAN. Information as a Service Based Architectural Solution for WSN. In *First IEEE Internation Conference on Communications in Chine: Advanced Internet and Cloud (AIC)* (2012).

[7] ASHTON, K. That ”Internet of Things” thing. *RFID Journal* (2009).

[8] BARNAGHI, P., HENSON, C. A., AND WANG, W. Semantics for the Internet of Things: Early Progress and Back to the Future. *Internation Journal on Semantic Web and Information Systems* (2012).

[9] BLACKSTOCK, M., AND LEA, R. IoT Mashups with WoTKit. In *IEEE International Conference on the Internet of Things (IoT)* (Wuxi-China, Oct. 2012).

[10] COMPTON, M., BARNAGHI, P., BERMUDEZ, L., GARCÍA-CASTRO, R., CORCHO, O., COX, S., JOHN GRAYBEAL, HAUSWIRTH, M., HENSON,

C., HERZOG, A., HUANG, V., JANOWICZ, K., KELSEY, W. D., PHUOC, D. L., LEFORT, L., LEGGIERI, M., NEUHAUS, H., NIKOLOV, A., PAGE, K., PASSANT, A., SHETH, A., AND TAYLOR, K. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Sicence, Services and Agents on the World Wide Web*, 17 (2012), 25–32.

[11] DINH, T., AND KIM, Y. An Efficient Sensor-cloud Interactive Model for On-demand Latency Requirement Garantee. In *IEEE Internation Conference on Communications (ICC), Paris, France* (May 2017).

[12] DISTEFANO, S., MERLINO, G., AND PULIAFITO, A. Sensing and Actuation as a Service: a new development for Clouds. In *11th International Symposium on Network Computing and Applications* (2012).

[13] DISTEFANO, S., MERLINO, G., AND PULIAFITO, A. A utility paradigm for IoT: The sensing Cloud. *Pervasive and Mobile Computing*, 20 (2014), 127–144.

[14] DUAN, Y., FU, G., ZHOU, N., SUN, X., NARENDRA, N. C., AND HU, B. Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends. In *8th IEEE International Conference on Cloud Computing* (2015).

[15] FORTINO, G., AND ET AL, C. S. *Towards Multi-layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach*. Springer International Publishing, 2018, ch. Integration, Interconnection and Interoperability of IoT Systems, Internet of Things, pp. 199–232.

[16] FORTINO, G., RUSSO, W., SVAGLIO, C., VIROLI, M., AND ZHOU, M. Modelling Opportunistic IoT Services in Open IoT Ecosystems. In *18Th Workshop "From Objects to Agents"* (Calbria, Italy, June 2017), pp. 90–95.

[17] GUERREIRO, J., RODRIGUES, L., AND CORREIA, N. Allocation of Resources in Se-aaS Clouds Managing Virtual Sensor Mashups. *To be Published*.

[18] GUERREIRO, J., RODRIGUES, L., AND CORREIA, N. On the Allocation of Resources in Sensor Clouds Under the Sensing as a Service Paradigm. *To be Published*.

[19] GUERREIRO, J., RODRIGUES, L., AND CORREIA, N. Fair Resource Assignment at Sensor Clouds Under the Sensing as a Service Paradigm. In *Technological Innovation for Resilient Systems* (Jan. 2018), vol. 521, DOCEIS, Springer, pp. 167–174.

[20] GUERREIRO, J., RODRIGUES, L., AND CORREIA, N. Modelling of Sensor Clouds Under the Sensing as a Service Paradigm. In *Broadband Communications, Networks and Systems* (Sept. 2018), Broadnets 2018.

[21] GUERREIRO, J., RODRIGUES, L., AND CORREIA, N. Resource Allocation Model for Sensor Clouds under the Sensing as a Service Paradigm. *Computers 8*, 1 (2019).

[22] GUINARD, D., AND TRIFA, V. *Building The Web of Things*. Manning Publications, 2016.

[23] GUINARD, D., TRIFA, V., PHAM, T., AND LIECHTI., O. Towards physical mashups in the web of things. In *IEEE Sixth International Conference on Networked Sensing Systems (INSS)* (2009).

[24] HSU, Y.-C., LIN, C.-H., AND CHEN, W.-T. Design of a Sensing Service Architecture for Internet of Things with Semantic Sensor Selection. In *In proceedings of the International Conference UTC-ATC-ScalCom, Bali, Indonesia* (2014).

[25] ISHI, Y., KAWAKAMI, T., YOSHIHISA, T., TERANISHI, Y., NAKAUCHI, K., AND NISHINAGA, N. Design and Implementation of Sensor Data Sharing Platform for Virtualized Wide Area sensor Networks. In *Internation Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Victoria, BC, Canada* (Nov. 2012), pp. 333–338.

[26] KANTI, S., AND BONNET, C. From the Internet of Thing to the Web of Things enabling Sensing as a Service. In *IEEE International Conference on Consumer Electronics – Taiwan (ICCE-TW)* (2018).

[27] KHAN, R., KHAN, S. U., ZAHEER, R., AND KHAN, S. Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges. In *10th International Conference on Frontiers of Information Technology* (2012), pp. 257–260.

[28] KIM, M., ASTHANA, M., BHARGAVA, S., IYYER, K. K., TANGADPALLIWAR, R., AND GAO, J. Developing an On-Demand Cloud-Based Sensing-as-a-Service System for Internet of Things. *Journal of Computer Networks and Communications* (June 2016), 1–17.

[29] KRČO, S., POKRIĆ, B., AND CARREZ, F. Designing IoT architecture(s): A European perspective. In *IEEE World Forum on Internet of Things (WF-IoT)* (2014), pp. 79–84.

[30] KUMAR, L. D., GRACE, S. S., KRISHNAN, A., MANIKANDAN, V., CHINRAJ, R., AND SUMALATHA, M. Data Filtering in Wireless Sensor Networks using Neural Networks for Storage in Cloud. In *International Conference ICRTIT, Chennai, Tamil Nadu, India* (2012), pp. 202–205.

[31] LAI, C.-F., CHAO, H.-C., LAI, Y.-X., AND WAN, J. Cloud-Assisted Real-Time Transrating for HTTP Live Streaming. *IEEE Wireless Communications 20* (June 2013), 62–70.

[32] LAI, C.-F., WANG, H., CHAO, H.-C., AND HAN, G. A Network and Device Aware QoS Approach for Cloud-Based Mobile Streaming. *IEEE Transactions on Multimedia 15*, 4 (June 2013), 747–750.

[33] MADRIA, S. Sensor Cloud: Sensing-as-a-Service Paradigm. In *19th IEEE International Conference on Mobile Data Management* (2018).

[34] MATHEW, S. S., ATIF, Y., AND EL-BARACHI, M. From the Internet of Things to The Web of Things - Enabling by Sensing as-a Service. In *12th International Conference on Innovations in Information Technology (IIT)* (2016).

[35] MATHEW, S. S., ATIF, Y., SHENG, Q. Z., AND MAAMAR, Z. Web of Things: Description, Discovery and Integration. In *International Conference on Internet of Things and Cyber, Physical and Social Computing (iThings/CPSCom)* (2011).

[36] MIHUI KIM, M. A. E. A. Developing an On-Demand Cloud-Based Sensing-as-a-Service System for Internet of Things. *Journal of Computer Networks and Communications* (2016).

[37] MISRA, S., BERA, S., MONDAL, A., TIRKEY, R., CHAO, H.-C., AND CHATTOPADHYAY, S. Optimal gateway selection in sensor-cloud framework for heath monitoring. *IET Wireless Sensor Systems. 4*, 2 (2014), 61–68.

[38] MISRA, S., CHATTERJEE, S., AND OBAIDAT, M. S. On Theoretical Modeling of Sensor Cloud: A Paradigm Shift from Wireless Sensor Network. *IEEE Systems Journal 11*, 2 (June 2017), 1084–1093.

[39] ONAT, F. A., AND STOJMENOVIC, I. Generating Random Graphs for Wireless Actuator Networks. In *IEEE International Symposioum World of Wireless, Mobile and Multimedia Networks* (Espoo, Finland, June 2007).

[40] PATIDAR, S., RANE, D., AND JAIN, P. Survey Paper on Cloud Computing. In *2nd IEEE International Conference on Advanced Computing & Communication Technologies* (2012).

[41] PERERA, C., ZASLAVSKY, A., CHISTEN, P., AND GEORGAKOPOULOS, D. Sensing as a Service model for Smart Cities supported by Internet of Things. *Transactions on Emerging Telecommunications Technologies, John Wiley and Sons, Inc. NY, USA 25*, 1 (September 2014), 81–93.

[42] PETROLO, R., LOSCRI, V., AND MITTON, N. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Transactions on Emerging Telecommunications Technologies, John Wiley and Sons, Ltd. 28*, e2931 (2017).

[43] PIYARE, R., PARK, S., MAENG, S. Y., AND PARK, S. H. Integrating wireless sensor network into cloud services for real-time data collection. In *International Conference on ICT Convergence (ICTC)* (2013).

[44] POURYAZDAN, M., KANTARCI, B., SOYATA, T., FOSCHINI, L., AND SONG, H. Quantifying User Reputation Scores, Data Trustworthiness, and User Incentives in Mobile Crowd-Sensing. *IEEE Access 5* (2017).

[45] SHENG, X., JIAN TANG, X. X., AND XUE, G. Sensing as a Service: Challenges, Solutions and Future Directions. *IEEE Sensors Journal 13*, 10 (October 2013), 3733–3739.

[46] UNION, I. T. *The Internet of Things*. ITU Internet Reports, 2005.

[47] WANG, W., WANG, Q., AND SOHRABY, K. Multimedia Sensing as a Service (MSaaS): Exploring Resource Saving Potentials of a Cloud-Edge IoT and Fogs. *IEEE Internet of Things Journal 4*, 2 (April 2017), 487–495.

[48] WU, M., LU, T.-J., LING, F.-Y., AND SUN, J. Research on the Architecture of Internet of Things. In *Advanced Computer Theory and Engineering (ICACTE)* (2010), vol. 5.

[49] XU, Y., AND MAO, S. A Survey of Mobile Cloud Computing for Rich Media Applications. *IEEE Wireless Communications 20* (June 2013), 46–53.

[50] YANG, Z., YUE, Y., YANG, Y., PENG, Y., WANG, X., AND LIU, W. Study and application on the architecture and key technologies for IOT. In *International Conference on Multimedia Technology* (2011).

[51] ZASLAVSKY, A., PERERA, C., AND GEORGAKOPOULOS, D. Sensing as a Service and Big Data. In *International Conference on Advances in Cloud Computing, Bengalore, India* (2012).

[52] ZHANG, D., ZHOU, Z., MUMTAZ, S., RODRIGUEZ, J., AND SATO, T. One Integrated Energy Efficiency Proposal for 5G IoT Communications. *IEEE Internet of Things Journal 3*, 6 (2016).

[53] ZHU, C., LI, X., JI, H., AND LEUNG, V. C. M. Towards Integration of Wireless Sensor Networks and Cloud Computing. In *IEEE 7th Internation Conference on Cloud Computing Technology and Science* (2015).

[54] ZORZI, M., GLUHAK, A., LANGE, S., AND BASSI, A. From today's INTRAnet of things to a future INTERnet of things: a wireless- and mobility-related view. *IEEE Wireless Communications 17*, 6 (2010), 44–51.

[55] ZUGE, A. P., AND CARMO, R. On comparing algorithms for the maximum clique problem. *Discrete Applied Mathematics 247* (2018).