TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABREVIATIONS

API	Application Programming Interface
AWS	Amazon web services
CPU	Central Processing Unit
DC	Data Center
EC2	Amazon Elastic Compute Cloud
GLPK	GNU Linear Programming Kit
GUI	Graphical User Interface
HVAC	Heating, Ventilation and Air-Conditioning
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service
I/O	Input/Output
IoT	Internet Of Things
ILP	Integer Linear Programming
IT	Information Technology

IP	Internet Protocol
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
LP	Linear Programming
MILP	Mixed Integer Linear Programming
NIST	National Institute of Standards & Technology OS Operating System
NP	Non-deterministic Polynomial-time
OS	Operating System
PaaS	Platform as a Service
PM	Physical Machine
QoS	Quality of Service
RAM	Random Access Memory
RAND	Research and development corporation
REST	Representational State Transfer

RTP	Real-Time Transport Protocol
RP	Raspberry Pi
SaaS	Software as a Service
SH	Smart Home
SLA	Service Level Agreement
SP	Service Provider
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine
VN	Virtual Network
VNE	Virtual Network Embedding
WiFi	Wireless Fidelity
XaaS	Everything as a Service

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

GB	Giga Bytes
GB/s	Giga Bytes per second
Ghz	Giga hertz
Mb	Mega bits
MB	Mega Bytes
ms	milli-seconds
s	seconds
TB	Tera Bytes
$/h	Dollar per hour

CHAPTER 1

INTRODUCTION

1.1 Context and motivation

The internet of things (IoT) industry is booming and businesses including equipment manufacturers, Internet providers, and service providers are perceiving future opportunities and are competing to provide the best IoT solutions to the market. By 2025, the IoT related industry profit is predicted to grow by 1.1 to 2.5 trillion per year (Whitmore, Agarwal et Da Xu, 2015). The sale of connected devices and services will amount to about 2.5 trillion dollars in 2020 (Whitmore, Agarwal et Da Xu, 2015). These studies confirm the “revolution” of the IoT industry and the great motivation toward it.

Generally speaking, IoT can be defined as a paradigm where everyday objects can be equipped with identifying, sensing, networking and processing capabilities that will allow them to communicate with one another and with other devices and services over the Internet (Whitmore, Agarwal et Da Xu, 2015).

One of the most emerging applications of the IoT is the smart home and home automation (Gubbi et al., 2013). The smart home concept promises to offer an easier and safer life as well as energy efficiency by means of automating households and minimizing user intervention in controlling home appliances and monitoring home settings. A smart home is typically equipped with sensors and cameras to measure home conditions such as temperature, humidity, luminosity and to control HVAC systems e.g. heating, ventilation and air conditioning in order to meet comfort and safety standards.

From the simple monitoring applications that control lighting, heating, and alarms to the video surveillance and face recognition ones, home automation applications are becoming more

sophisticated and demand more computing resources. For example, using a web camera to monitor a home, or its surroundings, can consume 20-40 % of central processing unit (CPU) resources of the home gateway (Igarashi et al., 2014). In addition, as a typical home gateway is quite costly, it limits thus the number of smart home end users and the expansion of the smart home industry. Another issue is that a home gateway is very difficult and expensive to be upgraded. This operation usually needs on-site technical intervention.

Applications running on a home gateway are resource-constrained thus making it really hard to host compute-intensive applications, in particular when several ones are running concurrently. This imposes limitations on both service provider and consumer. On one side, the service provider who has no previous knowledge of popular services finds himself limited in which applications can be supported and which should be dropped. On the other side, the end user finds himself stuck with a set of uncustomized services resulting in a lesser quality of user experience.

Cloud computing as it offers on-demand, pay-per-use and scalable computing resources (e.g. CPU, memory, storage) (Mell et Grance, 2011) is a promising solution to surpass the limitations in the future demand of smart home applications. Using cloud computing would allow the consumer to access, monitor and control home devices and appliances anytime and from anywhere. Migrating smart home vertical applications to the cloud can offer a better flexibility to the user to customize or update services and unlimited choice for the service provider to choose which applications to provide to the end user.

Therefore, cloud offloading of smart home applications has increasingly been adopted recently.(Padmavathi, 2016) Unlike traditional smart home applications which run only on a home element, cloud-based solutions have one or more components running locally connected to other components on the cloud and they jointly constitute an application fully accessible to the user.

Today, there are many cloud-based smart home services such as SmartThings Hub (Samsung, 2017) released by Samsung Electronics. This service supports third-party devices and

applications, and can be remotely controlled from mobile devices using different operating systems. For example, Nest, an IoT platform by Google, already provides cloud connectivity and device-to-device interaction, and control of IoT devices in Android. There is also an Amazon IoT (AWS, 2017b) which is an IoT platform responsible for connecting devices to amazon web services (AWS) compatible home devices (Derhamy et al., 2015).

Such solutions demonstrate encouraging results about the merging of cloud computing and smart home technologies. However, as far as we know, no existing solution has dealt with the application placement problem in the smart home context. Existing application placement solutions do not consider the smart home application-specific constraints such as providing the required bandwidth capacity between local-based components and cloud-based components and the interdependencies between the applications' components, which may result in deployed applications with poor performance. Moreover, most of the prior work only supports simple cost models which may result in sub-optimal solutions, especially in utility environments such as cloud computing where the pricing model is not linear according to the resource utilization.

Furthermore, existing cloud-based smart home solutions do not provide an automatic deployment of these complex services which will quickly become necessary for a smart home scenario where the same set of services are deployed for multiple users.

In alignment with the cloudification of smart home systems and the complex deployment of home applications, current smart home service providers require a solution to enable automatic deployment of its services onto cloud at minimal costs. The cloud provider has to provide such solution to smart home service providers, considering smart home specific requirements like minimizing the communication delay between home-based components and cloud-based components and meeting different types of capacity and application interdependency constraints while maximizing the utilization of its cloud infrastructure resources.

Consider a scenario as illustrated in Figure 1.1, where a smart home provider wants to deploy two applications in a set of homes. Let's say that these services are face recognition (represented by sky-blue nodes) and video monitoring (represented by navy blue nodes). The face recognition application is composed of a video/image capturing component which is located at home, and three other components, an image analysis component, a face recognition component and a database, which are operating in the cloud. The video monitoring application is composed likewise of a video/image transferring component which is located locally at the home, and four other components, a motion detection component, a video/image uploading component, and a user notification component, which are operating in the cloud.

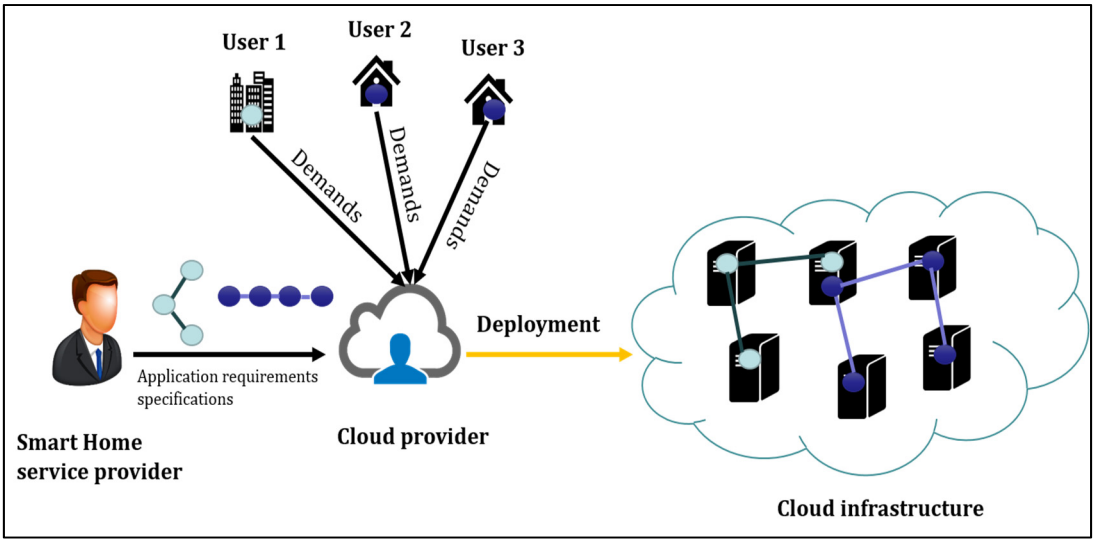

Figure 1.1 Scenario of complex service deployment

Optimally placing these application components onto shared cloud infrastructure at minimal costs while considering application specific requirements is known to be an NP-hard problem(Andersen, 2002). Moreover, manually deploying complex services onto cloud infrastructure is not a trivial task. To attempt to solve these two issues, we propose two separate contribution of this thesis:

1) A mixed integer linear programming (MILP) based algorithm, namely OptiDep to solve the application placement issue in a smart home context.

2) An application virtualization platform to enable the automatic deployment of complex services.

1.2 Problem statement

The cloud is considered to be an efficient solution to provide nearly unlimited resources to handle newly emerging home automation applications that can be accessible from anywhere. However, deploying home automation applications onto cloud faces many challenges. Offloading home automation applications to the cloud may cause additional network traffic overhead and a higher latency due to the distance between home-based and cloud-based application components, especially for interactive applications that are delay-sensitive. Moreover, over-provisioning cloud resources can result in additional costs, which sometimes can be very costly. Furthermore, manual deployment of complex services onto the cloud can be complex, time-consuming and error-prone. Therefore, cloud providers have to offer a service that ensures optimal provisioning and automatic deployment of the complex services.

One of the major issues in designing a platform to offer this service is solving the virtual-to-physical resource mapping. Resource mapping is a process that assigns existing resources to application components according to specific requirements.

The application requirements often include compute and network resources. Compute resources are the collection of processors, memory, and storage capacity required for an application component to run properly. The network resource is mainly bandwidth capacity needed to send data between application components. For example, a video streaming component that sends MPEG-2 flows to a video processing component requires at least 2Mbps bandwidth. The resource mapping process is known as the application placement problem which is a highly complex problem. Its solution requires to minimize the mapping costs, ensure the required performance of the deployed services and maximize the cloud computing and networking resource utilization.

Overall, four major challenges have to be considered when building an efficient and optimal virtualization system:

- ***P1: Cost***

Allocating more resources than required when virtualizing applications in the cloud will incur unnecessary costs especially when allocated resources are charged by cloud providers. In reality, computing and networking resources in the cloud are not priced linearly according to their processing power. In fact, cloud providers have employed different pricing models in order to charge for the utilization of consumed resources. Currently, the most popular one is the "pay-as-you-go" model where clients pay a fixed price per time unit. The world-leading cloud providers mostly adopt this pricing model, for example, Amazon (AWS, 2017a) and Microsoft with Windows Azure (Microsoft, 2017b). These cloud suppliers charge a fixed price per hour and per instance type. Another pricing model, which is widely adopted is the "subscription" model in which the client pays in advance for the resources he is going to use for a predefined time period. As for comparing between the three leading current cloud providers (e.g. Amazon Web services, Microsoft Azure and Google Cloud) in terms of cost minimization, it has been shown based on RightScale(RightScale) that, depending on the customer's needs, this latter will choose the suitable cloud provider. For example, if customers use a solid-state memory drive then Microsoft azure is the most cost-efficient option. Otherwise, Google may be considered as the best choice. AWS is usually considered as the middle-priced option among the three cloud providers. In general, according to RightScale, Google Cloud ensures the lowest cloud provider in terms of on-demand pricing for the VMs.

Our proposed solution must take into account nonlinear pricing models. It will be based on a commonly used pricing model currently adopted by cloud providers mentioned above to get accurate results.

- ***P2: Quality of Service***

Cost minimization may degrade the performance of applications. The challenge here is to provide the required quality of service (QoS) to clients' requests. For example, media applications for domestic entertainment require high-capacity and rigorous Quality-of-Service (QoS). Their compute-intensiveness will involve real-time interconnection of multiples, distributed and high-performing processing and storage resources. Offloading media applications to the cloud will impose additional network traffic overhead and incur additional delay that can result in a poor performance.

Therefore, our proposed solution must ensure QoS for smart home applications by providing the required bandwidth capacity to minimize the communication delay between local-based components and cloud-based components.

- ***P3: Automated deployment of home automation applications***

Since we are dealing with multi-component home automation applications, it is not possible to simply deploy the set of proper services on a single instance and try to just duplicate the image of an instance on several VMs in the cloud. In fact, the configuration of distributed applications needs additional information about the different instances hosting the various services e.g. IP addresses, protocols, etc. Moreover, distributed systems are often composed of dependent services which are ordered (e.g. used) in a certain hierarchy that has to be respected when configuring them. This problem is worsen when there is a need to deploy home automation applications at a larger scale. Manually configuring such complex deployments is complex, error-prone and time-consuming, particularly when it has a large number of interdependent modules.

- ***P4: Resource utilization***

Finally, allocating more cloud resources than needed results in idle and wasted capacities. A good application placement solution must consider maximizing the utilization of the available computing and networking capacities to take full advantage of the cloud infrastructure resources paid for.

1.3 Research questions

To address the four aforementioned challenges, the following key research questions have been raised:

- ***RQ1: How should we model smart home applications to optimally virtualize each application component in a cloud environment?***

  The proposed system modeling has to take into account the specific characteristics of smart home applications such as interdependency requirements, delay communication requirements and capacity requirements.

- ***RQ2: How can we efficiently map applications to cloud resources given the physical capacity constraints in order to meet QoS requirements and minimize costs?***

  The purpose is to design a resource mapping algorithm that allocates compute and networking resources at minimal costs and maximal resource utilization while meeting application QoS.

- ***RQ3: How can we automate the resource provisioning and application deployment process?***

  The system should provide an automatic configuration, deployment, and provisioning of applications. The proposed architecture should be later implemented and validated with different smart home applications.

1.4 Objectives

Our main objective, in this thesis, is to design a system that automates the optimal deployment of smart home applications while maximizing the resource utilization of the cloud infrastructure.

This main objective is divided into four sub-objectives, as follows:

- ***O1***: Building a model to represent smart home vertical applications and cloud resources;
- ***O2***: Building an optimization model for cost minimization while maintaining the required quality of service (QoS);
- ***O3***: Developing an algorithm to map applications' components to available resources while meeting applications' requirements;
- ***O4***: Designing an architecture to automate the resource provisioning and application deployment process onto cloud.

1.5 Plan

The present thesis is divided into five chapters organized as follows:

- The first chapter is a general introduction. We first present the general context and motivations of this research. Then, the problem statement, the related challenges and accordingly, the objectives to be achieved are presented.

- The second chapter discusses the technical background. It is divided into two parts. The first part presents a synthesis of cloud computing and virtualization concepts and the second part introduces the smart home context consisting of a review of existing cloud-based smart home solutions.

- The third chapter is centered on related work. It first presents a review of the prior research that has dealt with the application placement problem and, based on their findings, a synthesis has been done to compare the different existing approaches, their limitations and highlight the contributions in this thesis.

- The fourth chapter is dedicated to the methodology. According to the objectives of our thesis, the first part is dedicated to the system modeling, and the second part discusses the proposed optimization model. The original OptiDep algorithm is then presented to solve the optimization model. The fourth part presents the architecture of the platform that implements OptiDep to automatically deploy applications. The final part shows a high-level view of the proposed system including the decision and deployment modules.

- The fifth chapter presents at first the implementation of the proposed system and then discusses the experimental setup and simulation results.

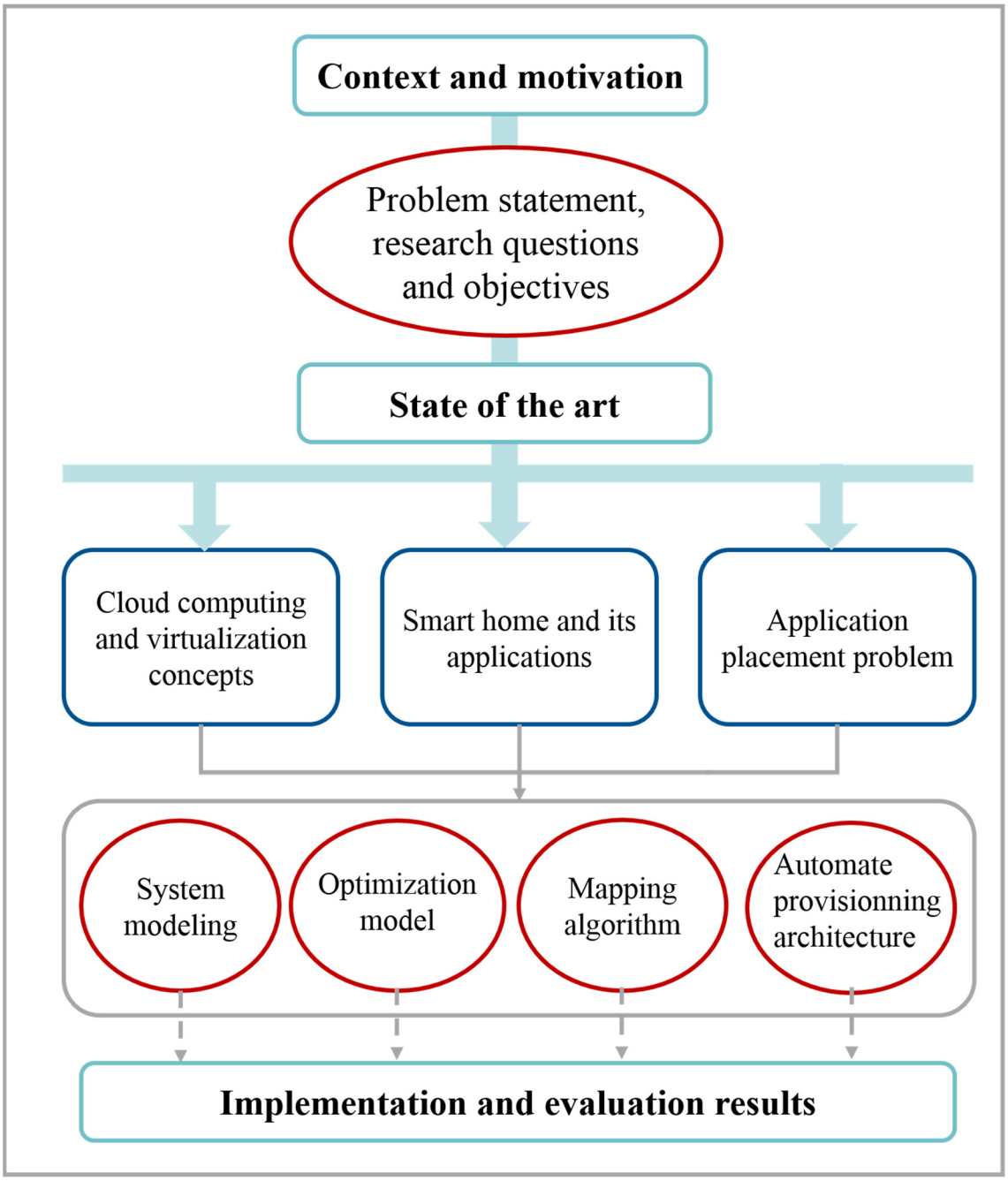

Figure 1.2 Thesis plan

CHAPTER 2

TECHNICAL BACKGROUND

This chapter presents the technical background of this thesis, including the concepts of cloud computing, virtualization technics, smart home and home automation applications.

2.1 Cloud computing and virtualization

Let us first have a look at the definition of cloud computing and virtualization concepts and present a view of their characteristics, types, and models to better understand our problem.

2.1.1 Cloud computing

2.1.1.1 Definition

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell et Grance, 2011).

Cloud Computing is characterized by five main characteristics:

- *On-demand self-service*: Cloud providers deliver resources whenever they are required to end users.

  One of the key features of cloud computing is that computing resources can be obtained and released on the fly. Compared to the traditional model that provisions resources according to peak demand, dynamic resource provisioning allows service providers to acquire resources based on the current demand (Zhang, Cheng et Boutaba, 2010).

- *Broad network access*: Cloud resources are accessible online from any location whenever there is a network connection.
- *Resource pooling*: The infrastructure provider offers a pool of computing resources that can be dynamically assigned to multiple resource consumers. Such dynamic resource assignment capability provides much flexibility to infrastructure providers for managing their own resource usage and operating costs (Zhang, Cheng et Boutaba, 2010).
- *Measured Service*: Cloud computing employs a pay-per-use pricing model. The exact pricing scheme may vary from service to service.

2.1.1.2 Models of Cloud Computing

a. Layered model

The Cloud computing architecture can be divided into four layers (Zhang, Cheng et Boutaba, 2010):

- *Application layer*: The application layer is the highest level of cloud computing architecture consisting of cloud applications;
- *Platform layer*: This layer consists of operating systems and application systems; The purpose of the platform layer is to minimize the burden of deploying applications directly into VM containers;
- *Infrastructure layer:* The cloud software infrastructure layer provides fundamental resources to other higher-level layers. Cloud services offered in this layer can be categorized into computational resources, data storage, and communications (Wolf, 2009). This layer also known as the virtualization layer creates a pool of storage and computing resources by partitioning the physical resources using virtualization technologies such as Xen, KVM, and VMware. The infrastructure layer is an

essential component of cloud computing, since many key features, such as dynamic resource assignment, are only made available through virtualization technologies;

- *Hardware:* The bottom layer of the cloud stack is responsible for managing physical resources of the cloud which are applied in data centers. Data centers are typically composed of racks of physical servers, routers, switches, power and cooling systems. Major issues at hardware layer include hardware configuration, fault tolerance, traffic management, power and cooling resource management.

The Cloud computing architecture, as mentioned above, is modular limiting cohesion and dependency between the different layers as shown in Figure 2.1.

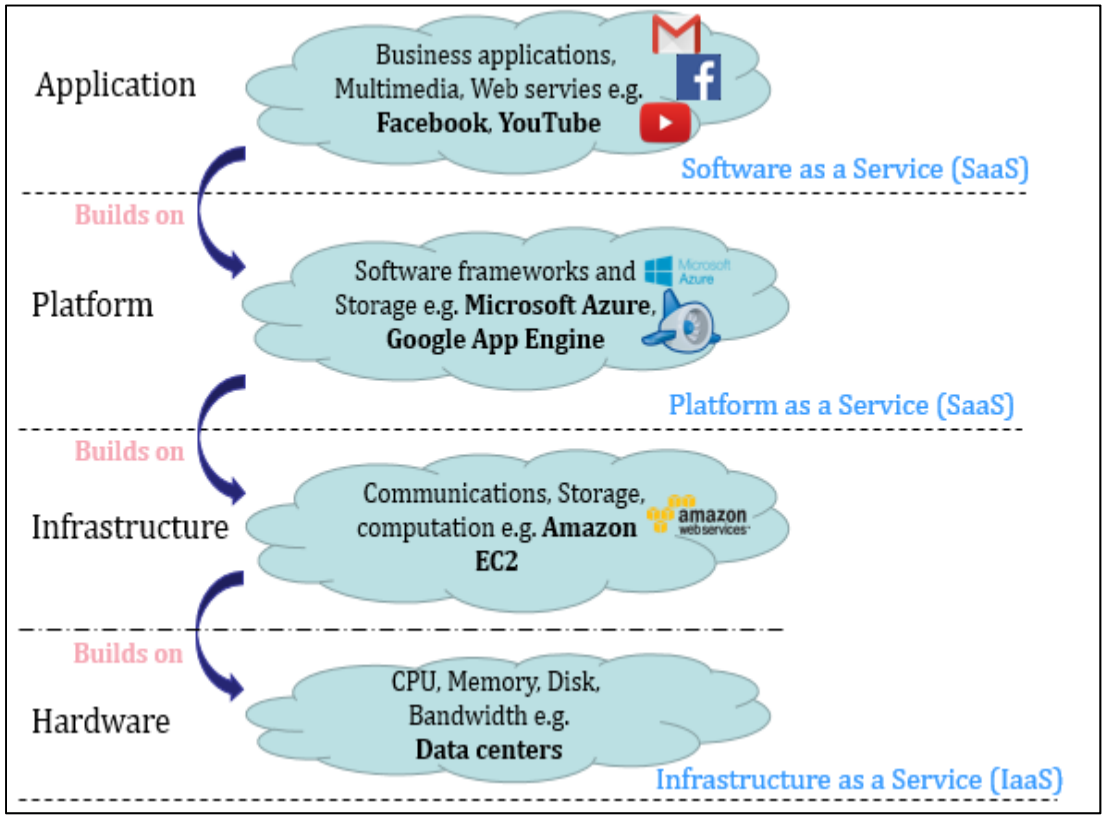

Figure 2.1 Cloud computing model

b. Business model

The business model of cloud computing consists of three main different layers. Each layer can be implemented as a service to the above one:

- *Software as a Service:* In the Software as a Service, an application like Gmail, Google docs, etc. is provided along with any software, operating system, network, and hardware;
- *Platform as a Service:* In the Platform as a Service, a network, an operating system is provided. Examples of PaaS providers include Google App Engine, Microsoft Windows Azure;
- *Infrastructure as a Service:* In the Infrastructure as a Service, only the hardware, and the network are provided. Examples of IaaS providers include Amazon EC2, Rackspace, etc.

2.1.1.3 Types of Cloud Computing

- *Public cloud*

In a public cloud, the whole computing infrastructure is located on the premises of a cloud computing company that offers the cloud service. The location remains, thus, separate from the customer and he has no physical control over the infrastructure. As public clouds use shared resources, they do excel mostly in performance, but are also most vulnerable to various attacks (Wolf, 2009);

- *Private cloud*

In this type of cloud, infrastructure (network) is used solely by a single customer or organization. The infrastructure is not shared with others, yet it is remotely located if the cloud is externally hosted. The companies have an option of choosing an on-premise

private cloud as well, which is more expensive, but they do have a physical control over the infrastructure. The security and control level is highest while using a private network. Yet, the cost reduction can be minimal, if the company needs to invest in an on-premise cloud infrastructure (Wolf, 2009);

- *Hybrid cloud*

A hybrid cloud combines public and private models to address drawbacks. A part of services are dedicated to private cloud and a part of them are offered to the public. Finding the best split between public and private components is important.

2.1.2 Virtualization

Virtualization can be viewed as the creation of a virtual version rather than the actual version of something, such as an operating system, network resources or a storage device where the system divides the resource into one or more execution environments (Rouse, 2016). Devices, end users and applications are able to interact with the virtual resource as if it were a real single logical resource.

Virtualization has numerous advantages. It allows a single physical machine to be shared among multiple instances securely and isolated from each other, enables dynamic resources provisioning and provides server consolidation facilities.

2.1.2.1 Types of virtualization

There are several types of virtualization:

- *Storage virtualization* is a sort of a grouping of physical storage from multiple network storage devices into one single storage device that is centrally managed (Moore et Baru, 2003). There are two types of storage virtualization which are bare-metal and hosted.

- *Network virtualization* is an approach consisting of grouping available resources in a network by splitting up the available bandwidth into channels, each of which is independent of others, and each of which can be assigned (or reassigned) to a particular server or device in real time. The main advantage of the network virtualization is that it divides the network into smaller parts easier to be managed.

- *Server virtualization* is the masking of server physical resources (including the number and identity of individual physical servers, processors, and operating systems) from server users. The main purpose behind this is to increase resource sharing and resource utilization while keeping the server resources details hidden to the user.

- *Application virtualization*

  In this thesis, we are focusing on application virtualization techniques.

  Application virtualization is the separation of the installation of an application from the client computer that is accessing it, as shown in Figure 2.2. The application continues to consider that it is still working normally, believing that it is still interacting with the operating system and uses the computer's resources as if the application has been installed directly on the operating system as normal. Thanks to virtualization, an application can be installed in a data center and preserved as an image to be delivered to the end users.

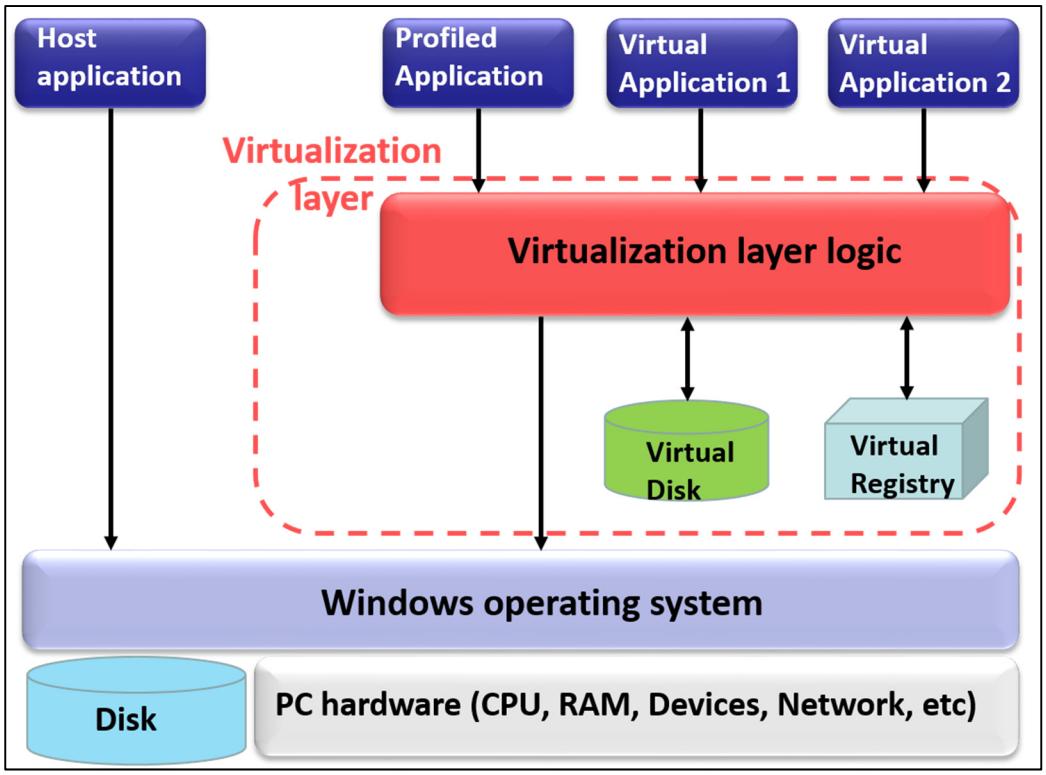

Figure 2.2 Application virtualization model (Cloud, 2013)

With this approach, it becomes then possible to deploy applications that were incompatible on the same piece of hardware since each application is isolated from other applications. This saves the time needed to test application compatibility. Though the application virtualization process has numerous advantages, there are many challenges to face:

- *Performance:* The main issue while virtualizing applications among application owners is performance. Under-provisioning applications will inevitably hurt performances and over provisioning will waste resources. Another point worth mentioning is that each application has its own requirements. Performance can be expressed in terms of CPU, memory, bandwidth, etc;

- *Supportability:* The supportability of applications on a virtual platform can be challenging. In fact, we should consider the vendor's support requirements for each application when virtualizing;
- *Management:* The loss of the ability to fully manage the application can be one of the main concerns to hesitate about virtualizing;
- *Reliability:* Application owners are looking for reliability. The fact that applications can remain online and operational is one of the most concerns for applications owners and businesses and can be an objection from them when virtualizing applications;
- *Security:* Another issue that is very important, particularly if the application is critical to the business, is security. Maintaining the security of an application while it is virtualized in the cloud can be challenging and must be considered as a high priority concern.

In this thesis, our focus is on performance, supportability and management challenges. Reliability and security are beyond the scope of this thesis.

2.2 Smart Home and home automation applications

A smart home is a home typically equipped with specially designed and structured wiring to enable occupants to remotely control or program an array of automated home electronic devices by entering a single command via home automation applications (Lee, Caytiles et Lee, 2013). Home automation applications can vary from the simplest lighting remote control to complex systems composed of networks of computers and microcontrollers for a high degree of automation. Smart home technologies can unlock both individual and society-wide benefits in different ways. They can provide financial savings, enhance convenience for consumers, contribute to more ecological and sustainable living, and reinforce the buyer's sense of safety and security (Lindsay, Woods et Corman, 2016).

2.2.1 Smart Home architecture system

The smart home architecture consists of a set of sensors and cameras belonging to a single vendor that are connected to a single home gateway through multiple means of networking e.g. Bluetooth, Zigbee, Wi-Fi, Z-wave, etc. All protocols for operating the set of connected devices are defined in the home gateway. The home gateway may control the device by itself or relay data to the vendor's application running on the cloud which will make decision for controlling VM devices. In case of local decision, the user may control the devices through a smart home application running on the smart phone which interacts with the home gateway.

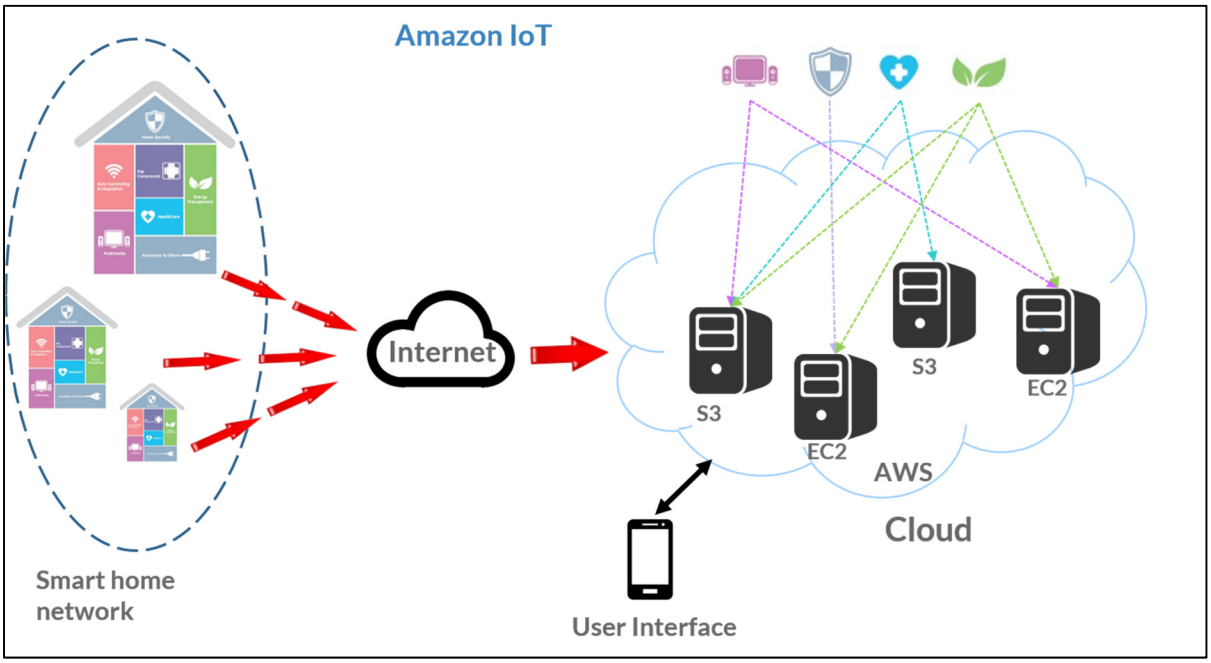

Figure 2.3 Smart home system architecture

Examples of home automation applications:

- A home surveillance application that notifies the user when there is a motion in his/her home;
- A door lock application that, using a face recognition module, opens the door automatically when the home owner arrives;

- A lighting monitoring application that automatically shutdowns the lights of a room if there is no motion detected in it for a time interval set by the user;
- A heart monitoring application that alerts the hospital in case there is a problem offering promising benefits to an elderly person living alone.

2.2.2 Smart Home existing solutions

Today, there are many smart home providers. The following section presents the most popular.

2.2.2.1 Amazon IoT

Amazon web services (AWS) IoT is a managed cloud platform that enables connected devices to interact with cloud applications and other devices. AWS IoT processes and routes messages to AWS endpoints and to other devices in a secure way. It allows end users applications to communicate with their devices(AWS, 2017b).

The architecture of Amazon IoT is shown in Figure 2.5. It is composed of:

- A device SDK to connect and authenticate the user's device. It also enables to exchange messages with AWS IoT using HTTP, MQTT protocols;
- A device gateway to enable devices to communicate with AWS IoT;
- Authentication and authorization module responsible for the authentication and the encryption of message exchanges between devices and AWS IoT;
- Registry module responsible for establishing a unique identity for devices;
- Device shadows to create a persistent, a virtual or a shadow version of each device that includes the device's latest state so that applications can read messages and interact with the device(AWS, 2017b);

- Rules engine is responsible for building IoT applications that monitor, process, analyze and act on data generated by connected devices. It also routes messages to AWS endpoints.

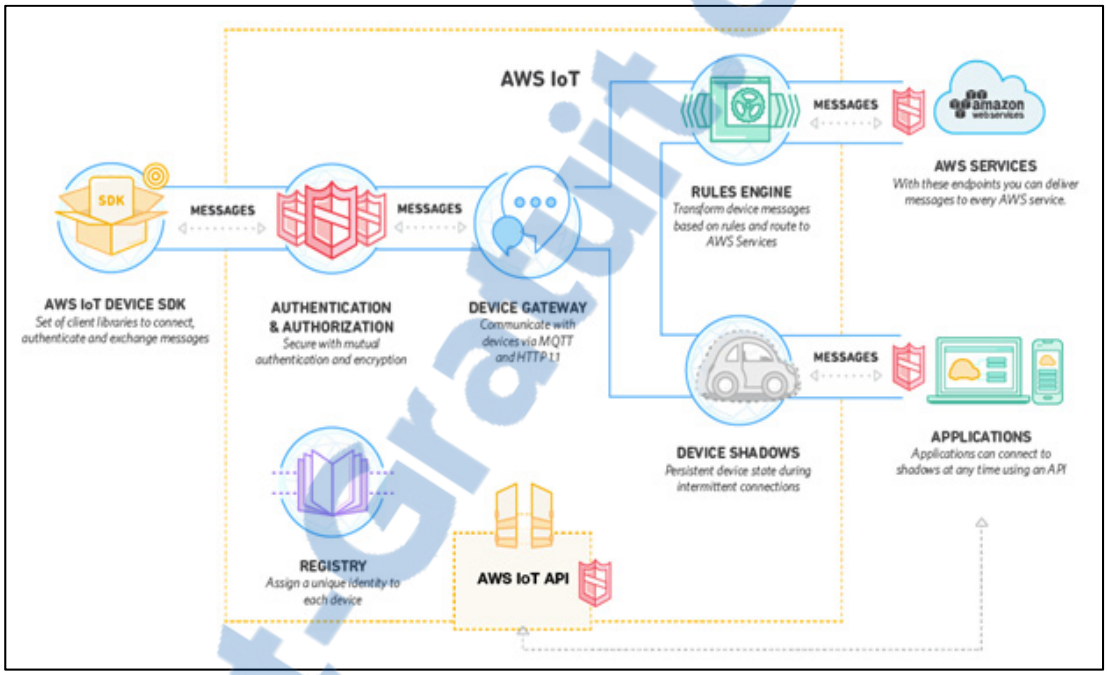

Figure 2.4 Amazon IoT platform overview (AWS, 2017b)

2.2.2.2 Azure IoT Hub

Azure IoT Hub(Microsoft, 2017a) is a service that enables bidirectional communication between devices and the business engine based in the Cloud as seen in Figure 2.6. The access is through authentication which is per-device using credentials and access control. Messages between devices and Cloud are bidirectional along the established channel.

Each device has two endpoints to interact with Azure IoT Hub: the first endpoint is from the device to the cloud where the device sends messages (e.g. telemetry data, request for execution, etc.) to the cloud, the second endpoint where the device receives a command for executing the requested action.

Azure IoT Hub also exposes two endpoints on the cloud side: the first endpoint is from the cloud to the device where the system can use this endpoint to send messages to the devices. This endpoint acts like a queue and each message has a TTL (Time To Live) after which it expires. The second endpoint is used to retrieve messages from the device.

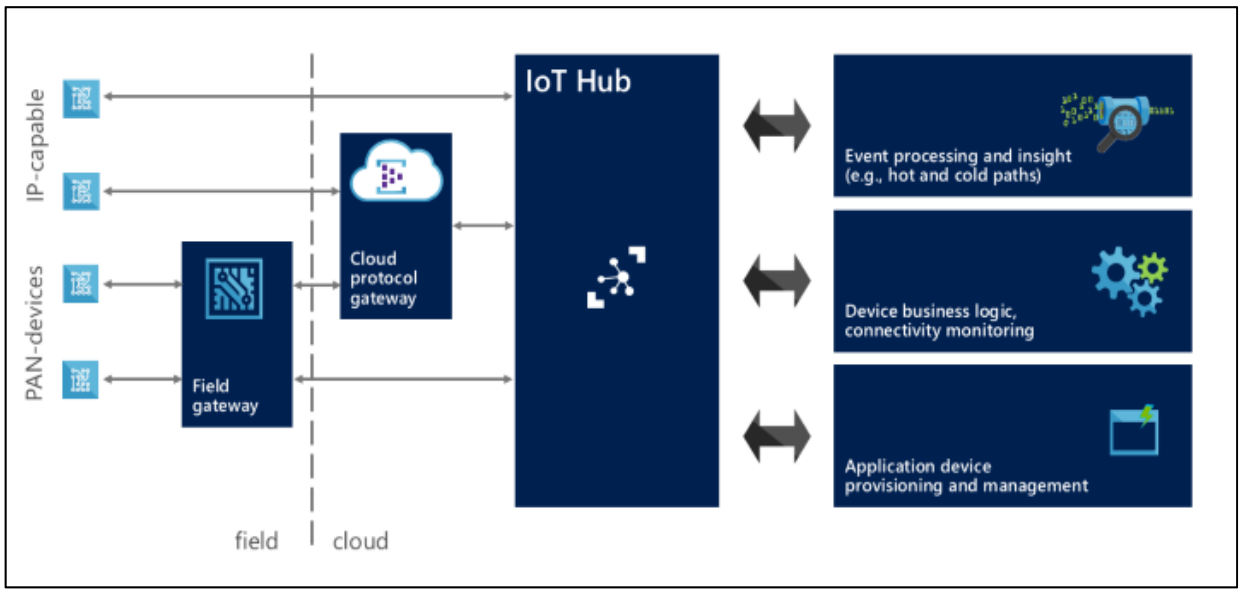

Figure 2.5 IoT architecture with IoT Hub (Patierno, 2015)

IoT Hub has an identity registry where it stores all information about provisioned devices. This information is related to identity and authentication. It provides monitoring information like connection status and last activity time; you are also able to enable and disable the devices using this registry. IoT Hub exposes another endpoint (device identity management) to create, retrieve, update and delete devices (Patierno, 2015).

2.2.3 Smart home applications requirements

Offloading applications to the cloud will bring many benefits such as easing the development and prototyping time with cloud platforms, providing flexibility and scalability, pricing savings, etc. However, smart home applications have specific requirements that have to be taken into account.

2.2.3.1 Heterogeneity

Hiding the heterogeneity of smart home devices coming from different smart home providers to offer a wide range of applications is required. This can be resolved by virtualizing smart home gateways for the different vendors and optimizing their placement on the cloud. This is outside the scope of our work.

2.2.3.2 Intra-application dependencies

Smart home applications may have feature interaction between two application components inside the same application. The performance will be degraded if these applications are deployed in distant virtual machines.

2.2.3.3 Increase in traffic demand

Communication between cloud-based components and local-based components incurs additional network traffic overhead. Besides, there is a challenge in QoS for different applications. For example, some streaming applications implement their own custom protocol like RTP and as network traffic is mostly TCP and UDP, this can cause a problem.

2.2.3.4 Timing and location

Home automation applications are characterized by specific constraints such as timing and location constraints. First, smart home applications affect the real world and thus the delay of transporting the data from the source to the sink must not exceed a certain threshold. Second, smart home applications interact with a set of sensors and devices placed at home and therefore, some application components must remain local. So, when being mapped, the distance between the local component and the remote component must be considered.

Conclusion

This chapter presented the technical background of this thesis. We have presented the concepts of cloud computing, virtualization concepts, smart home solutions and finally presented the specific requirements of smart home applications that we have to consider in our solution.

CHAPTER 3

LITERATURE REVIEW

In this chapter, we first review existing solutions related to the application placement problem. Accordingly, we analyze their main advantages and drawbacks and then highlight the novelty and contributions of our proposed approach.

3.1 Application placement problem

One of the major goals of cloud computing is to map applications to resources at minimal costs, e.g. to pay only for the resources that are really used. Existing solutions have used simple resource utilization indicators and they have not considered pricing concerns. On the other hand, there are also major challenges with performance requirements, especially with smart home specific constraints. In order to achieve this, we have to first solve **the application placement problem.**

Resource mapping is a system-building process that enables a community to identify existing resources and match those resources for a specific purpose. The process of mapping application components to cloud infrastructure resources influences the end user's quality of experience.
Application placement is the step of selecting the most optimal instances to host the set of application components given their computing and networking requirements.
An allocation which is directed by a decision system under user control can result in high resource supply costs. However, an allocation directed by a decision system under provider's control can result in low user-perceived resource value (Manvi et Shyam, 2014). A goal in application placement is to allocate the needed resources to the end user at minimal cost while maximizing the cloud infrastructure resource utilization.

3.1.1 Application placement algorithms

The application placement problem is reported to be an NP-hard (Andersen, 2002). Exact solutions optimally solve solutions but are not well adapted for large scales. Heuristic solutions are proposing an approach to solving problems in a practical manner without guaranteeing to be the optimal solution. The execution time of heuristic solutions is low compared to the exact approach. However, they focus on the local optimum that, in most cases, is far from the global optimum. Meta-heuristic solutions may have better results than heuristic solutions as they try to escape from the local optima to perform an almost acceptable search of solution space. In this research work, we propose an exact approach solution that optimally solves the application placement problem.

Depending on the type of principal approach used to attain the desirable mapping, we will divide the application placement existing work into exact approach, heuristic, and meta-heuristic solutions.

3.1.1.1 Exact approach

Exact solutions to the application placement problem can be achieved using integer linear programming (ILP) (Houidi, Louati et Zeghlache, 2008), (Yu et al., 2008), (Butt, Chowdhury et Boutaba, 2010).The integer linear programming (ILP) problem is a mathematical model where we maximize or minimize a linear function subject to linear constraints and in which some or all of the variables are integers.

Integer linear programming (ILP) can be used to model the application component mapping and the communication edge mapping. Several algorithms try to solve the problem such as branch and bound, branch and cut, etc. Several solvers support these algorithms e.g. GLPK or CPLEX (Meindl et Templ, 2012).

(Houidi et al., 2011) have addressed the virtual network allocation problem. To solve the problem, they have proposed an exact embedding algorithm that provides simultaneous node and link mappings in order to minimize the embedding cost for infrastructure providers while increasing the acceptance ratio of requests. For that, they have formulated the virtual network embedding problem as a mixed integer linear problem (MILP).

Authors have expressed the embedding cost of a virtual network request as the sum of costs of allocated infrastructure resources in regard to the demands of the virtual network requests which is expressed as follows:

$$Min\ (\sum_{e \in L_s} w_e \sum_{i,j \in N_v} f_e^{ij} + \sum_{i \in N_v} w_i t_i) \tag{3.1}$$

Where f_e^{ij} represents the amount of bandwidth assigned from the infrastructure link e to the virtual link between nodes i and j, t_i is the amount of bandwidth required at the virtual node i , w_e and w_i are uniformly distributed variables.
This proposal shows very encouraging results because it enables a simultaneous node and link mapping. However, in their objective function proposal, they have considered embedding cost as a linear function of the resource utilization which will result in suboptimal solutions mainly in utility environments where resources are not priced linearly to their processing power. Moreover, this solution has not considered different types of compute and network resources.

(Botero et al., 2012) have proposed an exact cost optimal solution to the virtual network embedding problem. For that, they have expressed the cost in terms of energy consumption. Their proposed solution consolidates resources and minimizes the set of mapped equipment in order to gain energy by turning off the inactive servers. Authors have used Mixed Integer Linear Programming (MILP) to solve the virtual network embedding problem.

Their objective function proposal aims to minimize the energy consumption by minimizing the set of inactive physical nodes and links that are activated after mapping a virtual network request. It is expressed as:

$$Min\ (\sum_{i \in V; NO_i = 0} \alpha_i + \sum_{(i,j) \in A; LO_{(i,j)} = 0} \rho(i,j)) \qquad (3.2)$$

α_i et $\rho(i,j)$ are binary variables indicating respectively whether the node i and the substrate link (i,j) are activated after the mapping.
This solution enables both node and link mapping and takes into consideration infrastructure specific constraints. However, their proposed solution differs from ours since they have expressed the cost in terms of energy consumption.

3.1.1.2 Heuristic

In cases where the computation time of an exact approach is not practical, heuristic-based approaches are adopted in order to achieve faster computation time needed. As we have discussed, heuristic solutions use a practical approach but are not guaranteed to be optimal. There is a great body of research work dealing with the application placement problem using proposed heuristic solutions.

(Chowdhury, Rahman et Boutaba, 2012) have suggested a virtual embedding solution that minimizes the embedding cost. This solution proposal coordinates better node and link mapping based on linear programming relaxation. It solves a mixed integer linear programming (MILP) problem and the multicommodity flow (MCF) problem through relaxation methods.

To do so, authors first perform the node mapping by introducing abstract nodes in the physical graph connected to a set of physical nodes for each virtual node. After that, they use the

multicommodity flow (MCF) problem to map the virtual links considering that each link is a connected to a pair of abstract nodes. The embedding problem is formulated with linear constraints on physical links and binary constraints on abstract links. The objective function is formulated as follows:

$$Min(\sum_{uv\in E^s}\frac{\alpha_{uv}}{R_E(u,v)+\delta}+\sum_{w\in N^s}\frac{\beta_w}{R_N(w)+\delta}\sum_{m\in N^{s'}/N^s}x_{mw}c(m)) \tag{3.3}$$

Where $R_E(u,v)$ and $R_N(w)$ are respectively the available capacity of a physical path and node, $\alpha_{uv}\in\{1,R_E(u,v)\}$ and $\beta_w\in\{1,R_N(w)\}$, f_{uv}^i represents the assigned flow on the physical edge uv for the virtual edge i and $c(m)$ is the CPU capacity of the node m.

This solution proposal has shown promising results compared to other mapping algorithms. However, their cost objective function is fully linear to the resource utilization. Moreover, though their solution consists of a better coordination between the node and link mapping, the two phases are still done separately resulting in sub-optimal solutions.

(Yu et al., 2008) have also researched the virtual network embedding problem. They have proposed the use of a greedy algorithm for the node mapping that greedily maximizes the resource utilization of the physical nodes. Then, they have considered two approaches for the link mapping, the unsplittable link mapping by adopting the k-shortest path algorithm and splittable link mapping by solving the multicommodity flow and problem. In the case where the multicommodity flow problem is unsolvable, the link mapping proposed algorithm reassigns the mapped nodes to the available ones. Their objective function aims to maximize the average revenue e.g. resource utilization and consists of:

$$lim_{T\to\infty}\frac{\sum_{t=0}^{T}R\left(G^v(t)\right)}{T}$$

$$R\left(G^v(t)\right)=\sum_{l^v\in L^v}bw(L^v)+\alpha\sum_{n^v\in G^v}CPU(n^v) \tag{3.4}$$

Where G^v represents the graph of the virtual network, $bw(L^v)$ is the bandwidth demand of the virtual link L^v and $CPU(n^v)$ is the CPU demand of the node n^v.
This solution proposal considers mapping nodes and links separately which will result in sub-optimal solutions. Moreover, similar to previous approaches, the cost model is expressed in terms of resource utilization.

In (Dubois et Casale, 2016), authors have proposed a heuristic approach that automates the application deployment decision while trying to minimize the spot prices and to maintain good performances. Authors have considered modeling applications as queuing networks of components. Their solution proposal consists first of choosing the minimum computational requirements for each application component. Next, it calculates the bidding price that minimizes the cost for each unit of rates and, based on it, decides which resources to rent and then considers the mapping of application components to the rented resources. Their optimization problem is formulated as follows:

$$\begin{aligned} & Min \sum_{y=1\ldots Y} C_y \\ & S.T. \\ & MRT_k(D) \leq \max MRT_k \ \forall k \\ & RTP_{u,k}(D) \leq \max RTP_{u,k} \ \forall u, \forall k \end{aligned} \tag{3.5}$$

The objective function aims to minimize the sum of rental prices such that the mean response time should be lower than their respective maximums. This solution proposal has shown promising results compared to other existing approaches. In addition, it has considered a pricing model adopted by the current Cloud providers which is not linear to the resource utilization. Nevertheless, this approach has only considered the node mapping in the formulation which leads to deployed applications with poor performance.

(Wang, Zafer et Leung, 2017) have proposed non-LP approximation algorithms to solve the application placement problem in the mobile edge-computing context. The authors first

considered the case of a linear application graph and proposed an algorithm for finding its optimal solution and then considered the tree application graph case and propose online approximation algorithms. This solution proposal has considered both node and link assignment in the application placement problem. Their optimization objective is based on load balancing.

$$\min \max\{max_{n,k}\, p_{n,k}(M)\,, max_l q_l(M)\} \tag{3.6}$$

$p_{n,k}(M)$ gives the total cost of the resource of type k requested by all application nodes that are assigned to node n and $q_l(M)$ is the total cost of all assigned edges. Their objective function is expressed linearly to the resource utilization.

This solution proposal is only limited to certain application topologies. Furthermore, the aim of the objective function is load balancing which is different from our approach.

(Lischka et Karl, 2009), authors have proposed a solution based on subgraph isomorphism that maps the node and link mapping at the same stage. The isomorphism solution is well defined in graph theory and is about finding a subgraph fulfilling the demands in the physical infrastructure. However, subgraph isomorphism method is known to output sub-optimal solutions in most cases.

3.1.1.3 Metaheuristic

Examples of metaheuristics solutions include genetic algorithms (Davis, 1991), ant colony optimization (Dorigo, Birattari et Stutzle, 2006) or tabu search (Glover et Laguna, 2013).

In (Pandey et al., 2010), a heuristic based on particle swarm optimization (Kennedy, 2011) is proposed to map application tasks to cloud resources while trying to minimize the rental costs. The proposed heuristic solution first calculates the computation and communication costs for all tasks and then uses a particle swarm optimization based algorithm to solve the task-mapping

problem. Though this solution has proven encouraging results compared to other heuristic-based solutions, its performance remains poor compared to an exact approach.

3.1.2 Comparison and discussion

3.1.2.1 Comparison

Regarding prior research, we have presented a brief summary of the most pertinent solutions to our research problem as described in Table 3.1. The following summary highlights the main differences between these solution proposals and our approach in terms of the nine following characteristics:

- NM: Considering the node mapping in the problem formulation.
- LM: Taking into account the link mapping of the problem formulation.
- CA: Proposing a solution that aims to minimize the mapping costs e.g. cost-aware.
- DF: Incorporating different capacities and networking requirements in the problem formulation.
- SNL: Suggesting an approach that enables a simultaneous node and link mapping.
- PM: Proposing a pricing model that takes into account the actual prices of the current Cloud providers.
- SH: Taking into account the smart home application-specific constraints such as minimizing the communication delay between local-based components and cloud-based components.
- IA: Considering interdependencies between application components in the solution.
- CI: Taking into account cloud infrastructure specific constraints e.g. compute and network constraints.

Table 3.1 Comparison of characteristics of related work

Approaches	NM	LM	CA	DF	SNL	PM	SH	IA	CI
(Yu et al., 2008)	✓	✓	✓	✗	✗	✗	✗	✗	✗
(Lischka et Karl, 2009)	✓	✓	✓	✓	✓	✗	✗	✗	✗
(Houidi et al., 2011)	✓	✓	✓	✗	✓	✗	✗	✗	✗
(Botero et al., 2012)	✓	✓	✗	✗	✓	✗	✗	✗	✓
(Chowdhury, Rahman et Boutaba, 2012)	✓	✓	✓	✗	✗	✗	✗	✗	✗
(Dubois et Casale, 2016)	✓	✗	✓	✓	✗	✓	✗	✓	✓
(Wang, Zafer et Leung, 2017)	✓	✓	✗	✓	✗	✗	✗	✓	✓
Our approach	✓	✓	✓	✓	✓	✓	✓	✓	✓

3.1.2.2 Discussion

The review of related work has led us to the following conclusions:

- The placement problem has been widely addressed in the field of network virtualization, coined as the virtual network embedding problem. However, there is very few research on the application placement problem. Prior research on this problem is mainly heuristic-based that do not consider simultaneous node and link mapping;

- Most of the prior research that has considered mapping costs as their objective function does not adopt the current pricing model offered by cloud providers in today's market. They simply considered a linear cost model for resource utilization;

- Existing solutions that considered current pricing models in their works are mostly heuristic-based algorithms that consider only node mapping resulting in sub-optimal solutions;
- As seen in chapter 2, cloud offloading of home automation applications is gaining interest in the research field, however, as far as we know, no existing solution has considered the application placement problem in the specific smart home context. The problem has mainly been considered in other contexts, like mobile computing. However, home applications are fundamentally different from mobile applications since they are not as interactive as mobile applications, e.g. a gaming mobile application may require a lot of interactions with the user as opposed to a monitoring application that gathers data from sensors, cameras... and then analyzes this data and sometimes reacts to it. Therefore, the application placement problem differs from the mobile context to the smart home context.

The main contributions of our proposed solution are:

- A mathematical optimization model that increases considerably the cost savings without incurring performance degradation by scheduling applications on their cost optimal instances and maximizing the cloud resources' utilization. The proposed solution is an exact approach that enables simultaneous node and link mapping and incorporates multiple types of compute and network resources.

- The proposed approach enables the cloud provider to find at first a feasible solution that meets the capacity constraints and second a solution to smart home application providers at a very concurrent price in the market while maximizing its resource utilization.

- An optimal algorithm for placing applications to solve the mathematical optimization problem and is, as far as we know, the first solution that takes into consideration specific requirements of smart home applications;

- The pricing model that we have adopted for evaluation results is based on actual prices of a cloud provider, which is not a simple pricing model linearly proportional to allocated resources.

Conclusion

In this chapter, we have first described the application placement problem. Second, we have presented existing solutions that have tried to address this problem. Finally, a comparative study and conclusions were presented to highlight the planned contributions of the proposed solution with regard to limitations of the existing work.

CHAPTER 4

METHODOLOGY

In this section, we present the experimental methodology of this research project. To that end, first, the requirements of the application virtualization platform are presented. Then, we describe the different steps that were executed in order to design and develop this platform. First, a system model is designed followed by an optimization model that optimally maps application components to cloud resources using our proposed algorithm. Finally, an architectural design was created with the objective to automate the application deployment process.

4.1 Application virtualization platform requirements

4.1.1 R1: Modeling Smart Home applications

Multi-component applications often consist of many services that depend on one another. In fact, a service may call some functions of another service or use its output. In order to optimally virtualize these applications, we have to respect the interdependencies. This means that the different nodes must be deployed in the appropriate order to respect the hierarchy of these dependencies. To achieve that goal, we have to properly model these applications. Some previous work has assumed a fixed architecture consisting of a master node and a collection of slave nodes. This severely limits the type of applications to be deployed. Our proposed system should support complex dependencies and enable nodes to advertise values that can be queried to configure dependent nodes.

4.1.2 R2: Efficient mapping of application components to Cloud resources

When monitoring cloud services, it has been seen that many services only need a small part of the resources allocated to them. In other words, several VM instances operate and consume much less than expected, resulting in a waste of resource and rising costs. Since a service provider wants to deploy his services at minimal costs and the cloud provider wants to maximize its resource allocation, a mapping mechanism must be set up to allocate only the needed resources. This can result in noticeable benefits such as minimizing costs, maximizing resource utilization, improving system availability and reducing infrastructure complexity.

4.1.3 R3: A mapping approach that maintains the required QoS

Trying to maximize resource utilization while mapping application components to cloud resources can result in resource under-provisioning and QoS degradation. This will inevitably hurt the performance of the deployed services. Therefore, it is important to develop a mapping algorithm that maintains the required QoS by responding to computing and networking requirements of services to be deployed.

4.1.4 R4: Automatic deployment of distributed applications

Smart home's distributed applications often need complex configurations and setup to be correctly installed. Therefore, deploying such services can be a challenging task mainly for the smart home service provider when these applications need to be deployed for a large number of homes. This can be time-consuming, error-prone and expensive since it may involve the repetition of many complex tasks. In order to save time and reduce errors, these complex repeated tasks should be automated so that a user can easily describe the services he needs, and then, according to that, these services are automatically deployed.

4.2 System modeling

In this section, we address the objective O1 which is about building a model to represent smart home vertical applications and cloud resources by proposing a system modeling that represents specific interdependencies between the different components of an application and constraints of cloud resources. Our proposed system will be composed of applications, virtual resources, and physical resources. We make the following assumptions: 1) that our system is stationary; and 2) that there is a limited number of available VM types e.g. flavor.

4.2.1 Application layer model

We model the application as a directed graph denoted as $G^c = (V^c, E^c)$, where V^c is the set of application components and E^c is the set of dependencies between application components. A dependency $e^c_{(i,i')}$ is explained by the fact that two components v^c_i and $v^c_{i'}$ are communicating in order to accomplish a certain task. For example, a video streaming component sending streaming flows to a video processing component to be analyzed requires 5 GB per hour.

Each application component v^c_i has capacity attributes e.g. minimum compute capacity $\alpha_{i,t}$, $t \in \{1,2\}$ 1: CPU , 2: RAM as well as a set of non-capacity attributes (e.g. OS type, location) and each dependency $e^c_{(i,i')}$ between two application components v^c_i and $v^c_{i'}$ also has capacity attributes e.g. minimum networking capacity in terms of bandwidth $\delta_{(i,i')}$ as well as non-capacity attributes (e.g. link type, QoS).

4.2.1.1 Resource requirements model

The application graph enables us to have a detailed view of the different dependent application components with their compute and network requirements. However, in practice, it is not always straightforward for users to input the “right” compute and network requirements, especially when the application models are complex, and the required resources depend on

other parameters e.g. QoS class, the number of users, etc. The difficulty lies in the fact that such dependencies are not made explicit in today's systems, therefore requiring the task of discovering these dependencies. What is needed is a model to find out the dependency relationships between compute and network requirements and parameters such as QoS class and number of requests. A technique which is very successful in modelling dependencies is statistical regression analysis(Mosteller et Tukey, 1977).

Statistical regression analysis on collected data on the output metric enables to fit regression lines indicating the presence and the strength of dependencies of the output QoS metric on the components that have been monitored. An advantage of the technique is its ability to differentiate causal relationships indicating actual resource dependencies from simple correlations in monitoring data since there is knowledge of which application component is being monitored. This technique is considerably successful in modeling dependencies. To that end, we have proposed an algorithm based on regression analysis to model dependencies between compute and network requirements and QoS class to help the user input its specifications.

Algorithm 4.1 Building application dependency models

Building application dependency models

Input: application components$\{v_i^c\}_{1\leq i\leq I}$

Output: Dependency models $\alpha_{i,1} = g$(QoS class), $\alpha_{i,2} = g$(QoS class), $\delta_{(i,i')} = g$(QoS class)

```
1.  for all v_i^c ∈ V^c
2.      for all v_i'^c ∈ V^c, i' ≠ i
3.          for all QoS classes
4.              Assess compute (e.g. CPU, memory) and network
                requirements (e.g. Bandwidth);
5.              for each requirement
6.                  Apply regression algorithms to model the dependency;
7.              end for
8.          end for
9.      end for
10. end for
```

Algorithm 4.1 takes as input the set of components of the application $\{v_i^c\}_{1\leq i\leq I}$ and outputs the dependency models. The algorithm first goes through all existing pairs of components $(v_i^c, v_{i'}^c)$ with $i' \neq i$ and for each QoS class, assess the compute and network requirements between the two components v_i^c and $v_{i'}^c$. After that, different statistical regression algorithms such as linear, polynomial, exponential and logarithmic algorithms are called to choose the best algorithm that models the dependency based on metrics like R-squared and adjusted R-squared.

4.2.1.2 Illustrative example

Let us consider an example of a video monitoring application that helps the user to remotely monitor kids, disabled or old persons in his house. The application is composed as shown in

Figure 4.1 of five components where arrows represent the interdependencies between application components. First, there is an IP camera connected to a video/image-transferring module responsible for sending the video/image stream. In the cloud, we find the motion detection module responsible for detecting any motion when processing videos/images received. Whenever a motion is detected, the video/image stream is saved and then uploaded to a web server for later visualization. The user notification component notifies the user of motion detected in his home. In this application, the motion detection component and the video/image databases are stored on the cloud because of the limited resources at home network.

To illustrate the resource requirements' model, the bandwidth usage between the locally-based video/image transferring module and the cloud-based motion detection module for example is increasing exponentially with the QoS; in this case, exponential regression algorithms may be the most appropriate algorithm to model the dependency. The bandwidth usage between the motion detection module and the video/image saving module is bursty; for that, we can use other machine learning techniques to model the bandwidth behavior for different data exchanges.

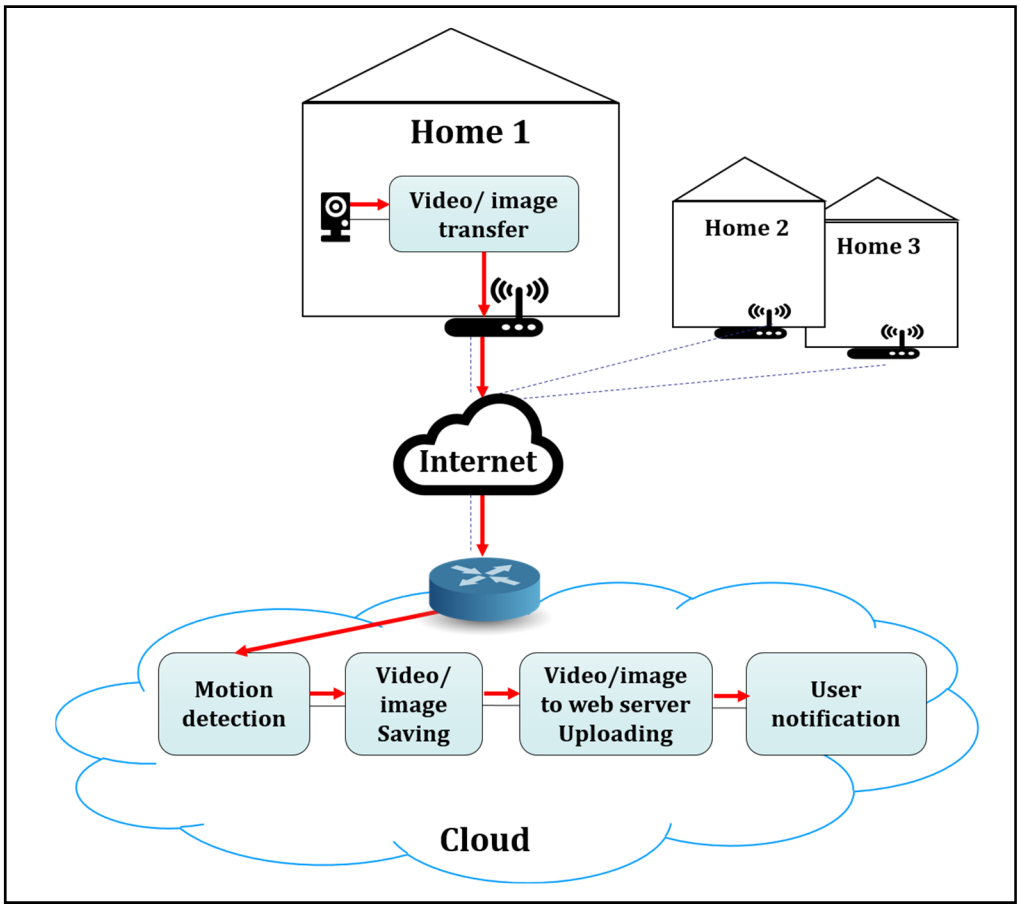

Figure 4.1 Scenario with video monitoring application

4.2.2 Infrastructure layer model

Cloud infrastructure can be modeled as an undirected substrate graph denoted as $G^s = (V^s, E^s)$.

Each physical server v_k^s has a set of capacity attributes e.g. available capacities $c_t(v_k^s)$, $t \in \{1,2\}$, 1: CPU, 2: Memory and a set of non-capacity attributes e.g. availability, location, processor type etc. Each edge $e^s_{(k,k')}$ between a pair of physical servers v_k^s and $v_{k'}^s$ has also a set of capacity attributes e.g. available bandwidth capacity b($e^s_{(k,k')}$) as well as non-capacity attributes e.g. QoS parameters, link type.

4.2.3 Virtual layer model

The virtual layer is built on top of the infrastructure layer according to the cloud infrastructure available capacities. It consists of virtual machines (VMs). It can be modeled as an undirected graph $G^v = (V^v, E^v)$ where V^v is the set of VMs and $E^v$is the set of virtual links between the VMs. Each VM type v_j^v has a predefined capacity $\beta_{j,t}$,$t \in \{1,2\}$, , 1: CPU, 2: Memory. Each application component v_i^c can be deployed on the VM instance v_j^v at a cost $\eta(v_j^v)$ depending on its characteristics (e.g. CPU, RAM, storage, etc).

An edge $e^v_{(j,j')}$ is the available bandwidth between two connected VMs v_j^v and $v_{j'}^v$. It has a capacity $\gamma_{(j,j')}$and a cost $\eta(e^v)$ per used resource (per GB bandwidth).

The following table 4.1 presents the parameters of the system.

Table 4.1 System parameters

I	Number of application components
J	Number of virtual machines
N	Number of physical servers
$\alpha_{i,t}$	Computing capacity of the application component v_i^c in terms of CPU and memory

Table 4.1 System parameters (continued)

$\delta_{(i,i')}$	Networking capacity of the dependency link $e^{c}_{(i,i')}$
$\beta_{j,t}$	Computing capacity of the virtual node v_j^v in terms of CPU and memory
$\gamma_{(j,j')}$	Bandwidth capacity of the virtual link $e^{v}_{(j,j')}$
$c_t(v_k^s)$	Compute capacity of the physical server v_k^s in terms of CPU and memory
$\mathrm{b}(e^{s}_{(k,k')})$	Network capacity of physical edge $\mathrm{b}(e^{s}_{(k,k')})$
$A = [a_{ij}]$	A binary matrix to represent mapping from an application component v_i^c to a virtual machine v_j^v
$B = [b^{(j,j')}_{(j_1,j_1')}]$	$b^{(j,j')}_{(j_1,j_1')}$denotes the flow mapped from virtual node $v^v_{j_1}$ to the virtual node $v^v_{j_1'}$ that passes through the virtual link $e^{v}_{(j,j')}$, $b^{(j,j')}_{(j_1,j_1')} > 0$
$X = [x_j]$	A binary matrix to represent a mapping to the virtual machine v_j^v.
$Y = [y_{(j,j')}]$	A binary matrix to represent a mapping to the virtual communication edge $e^{v}_{(j,j')}$
$z^{(i,i')}_{(j,j')}$	is a binary variable equal to $a_{ij}.a_{i'j'}$.
$bw(j,j')$	is the amount of bandwidth allocated from virtual node $v_j^v$to virtual node $v_{j'}^v$ that will support the demand of one or more dependency links $e^{c}_{(i,i')}$
$\mu(.)$	Mapping function
$\eta(.)$	Rental costs
$F(.)$	Cost function

4.3 Resource provisioning

As we have seen, the cloud provider is responsible for provisioning resources to the smart home provider in order to deploy its applications onto the cloud.
Upon receiving a request, the cloud provider identifies among the cloud physical servers the candidate virtual machines able to match the requested application required capacities by applying the capacity attributes. According to that, the mapping process is about selecting the set of virtual machines and edges that minimizes the overall cost while satisfying the compute and network demands.
The resource provisioning includes both the resource matching and the resource mapping steps.

4.3.1 Resource matching

This step is based on the selection of candidate virtual nodes that are able to support the applications is based on the capacity requirements. Let $Match(G^s) = \{v^v \in G^v\}$ denotes the set of candidate virtual machines able to host the requested applications. The aim of the Cloud provider is to define for each incoming request the $Match(G^s)$.
The matching process reduces the search space to make the resource mapping step faster.

4.3.2 Resource mapping

The cloud provider is also responsible for mapping applications to the set of candidate virtual graphs. Resource mapping consists of selecting for each application component and each dependency link the cost optimal virtual node and virtual paths that ensure optimal resource mapping. In order to maximize the resource utilization, we have considered VM consolidation and link splitting in our mathematical model. The aim of our proposal is to propose an exact embedding algorithm where node and link mapping stages are simultaneously executed.
To this effect, we define a mapping function μ: $G^c \rightarrow G^v$ such that:

$$\mu(v_i^c) = v_j^v \in V^v \tag{4.1}$$
$$\mu(e^c) = \mu(v_i^c, v_{i'}^c) = P^v\left(\mu(v_i^c), \mu(v_{i'}^c)\right) \in E^v$$

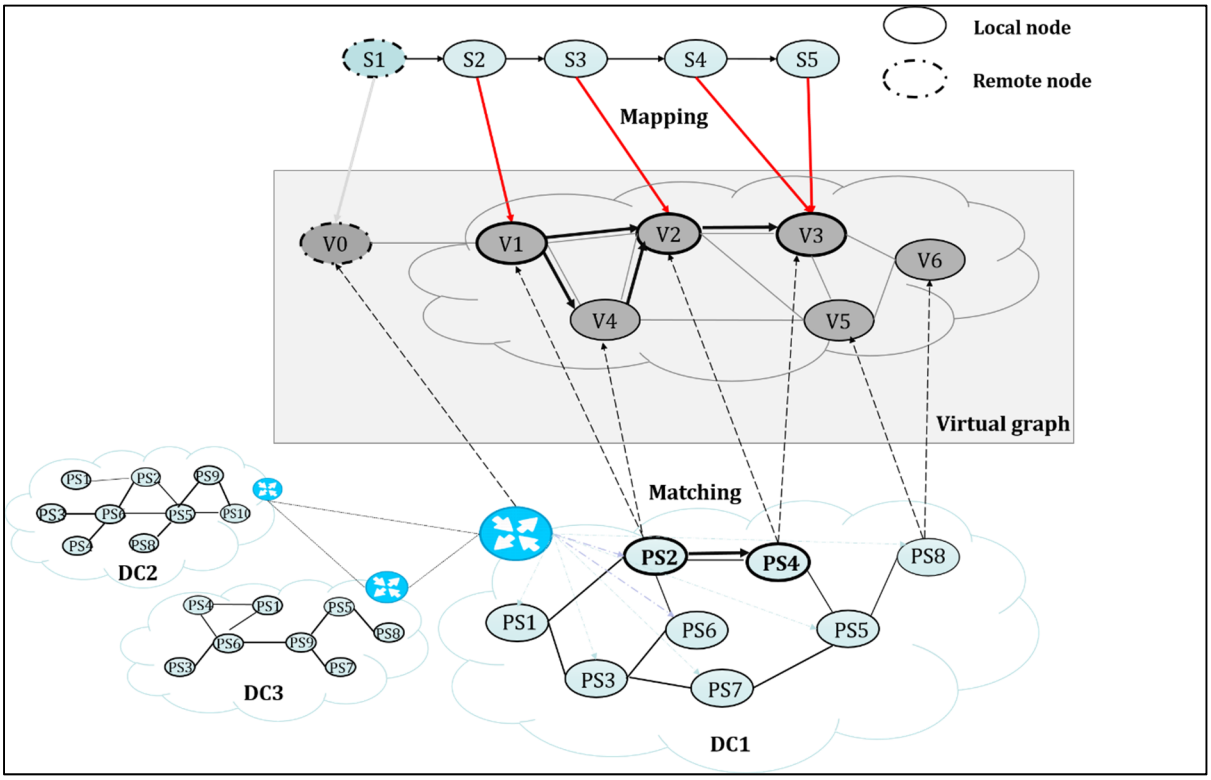

Figure 4.2 Application placement problem

The video monitoring application presented in Figure 4.1 can be represented as a linear chain of 5 services as shown in Figure 4.2. The first service is locally constrained e.g. it cannot be migrated to the Cloud. It can be abstracted as an application node with a null capacity $\alpha_{i,t} = 0\,, t\,\epsilon\,\{1,2\}$.

The other services S2, S3, S4, S5 (e.g. motion detection, video/images saving, video/images uploading to the web server and user notification) are deployed in a cloud environment. V0 is a hypothetical node in the virtual graph with a null capacity mapped to the local application component. During the matching process, a virtual graph has been built on top of the infrastructure graph depending on the physical capacity and the application requirements.

Possible mappings exist in three data centers DC1, DC2, and DC3 in three different locations. However, DC1 is selected as the optimal location during the mapping process.

In Figure 4.2, we show an example of optimal mapping. For instance, Service 2 is mapped to the virtual machine V1 because it is the one that satisfies its capacity requirement. Service 3 has two potential virtual machines that satisfy the capacity constraints V2 and V4, it is mapped to the service V2 because it is the most cost-optimal virtual machine. Service 4 and service 5 are consolidated on the same virtual machine V3 ({S4, S5} →V3) because it minimizes costs and maximizes the resource utilization.

Considering the dependency links, we remark that the shortest path for the dependency link (S2, S3) is (V1, V2). Nevertheless, (S2, S3) is split into two paths (V1, V2) and {(V1, V4);(V4, V2)} because the edge (V1, V2) does not have the required bandwidth capacity.

4.4 Mapping costs of Cloud resources

We have adopted a cost model in which the application provider is charged per type of mapped resources and per time unit. In our model, each allocated virtual machine instance has a rental cost $\eta(v^v)$ and each allocated edge between two virtual machines has a rental cost $\eta(e^v)$. Our work is inspired by amazon cost model but there are additional existing cost models which are being used by other cloud providers.

The mapping cost is calculated by summing up all the costs of mapped Cloud resources.

$$F(\mu)=\sum_{\mu(v^c)\in V^v}\eta(\mu(v^c)) \; + \; \sum_{\mu(e^c)\in E^v}\eta(\mu(e^c)) \qquad (4.2)$$

The cost of mapping the application graph onto cloud resources is calculated by summing up the rental costs of all the mapped nodes and edges.

Suppose that services 2 requires 1 CPU and 1 GB and service 3 requires 2 CPU and 0.9 GB of memory, Service 4 requires 3 CPU, 2 GB of memory and service 5 requires 1 CPU and 0.5 GB of memory to function properly. To simplify, we assume that all links between the components are 10 GB/h with a cost of 0.08 per GB per hour.

Suppose that the cost of a small instance (1 CPU, 2 GB) hosting the service S2 is 0.04$/h, the cost of a storage instance (2 CPU, 3.75 GB) hosting the service S3 is 0.5$/h, the cost of a large instance (4 CPU, 8 GB) hosting services 4 and 5 is 0.3 $/h.
The overall mapping cost is calculated as follows:

$$F(\mu) = 1 * 0.04 + 1 * 0.5 + 1 * 0.3 + 0.08 * 10 * 4 = 4.04\ \$/h$$

4.5 Problem formulation

In this section, we address the objective O2 to build an optimization model based on cost minimization while maintaining the required performance.

Our goal is to decide which cloud resources fulfill demands at minimal costs. In order to maximize the resource utilization, we assume that a single virtual machine can host one or more application components and that directly connected adjacent application components can be deployed in non-adjacent instances. We also consider the splittable flow scenario e.g. an application dependency while being mapped can be split into one or many networking edges. $z_{(j,j')}^{(i,i')}$ is an auxiliary binary variable equal to $a_{ij}.a_{i'j'}$ introduced to avoid the non-linearity of the formulation (see (Houidi et al., 2011)) and $bw(j,j')$ is the amount of bandwidth allocated from virtual node v_j^v to virtual node $v_{j'}^v$ in order to support network requirements of one or more dependency links $e_{(i,i')}^c$ such that:

$$\sum_{e_{(i,i')}^c \epsilon E^c} \delta_{(i,i')}\, z_{(j,j')}^{(i,i')} = bw(j,j')\ \forall\ v_j^v, v_{j'}^v\ \in\ V^v \qquad (4.3)$$

Each application node is allocated to exactly one virtual machine. This is expressed in the following constraint (4.4).

$$\sum_{v_j^v \epsilon V^v} a_{ij} = 1\ \forall\ v_i^c\ \in\ V^c \qquad (4.4)$$

The mathematical model should ensure that the compute demands are provided and that the compute cloud resources are not violated.

$$\sum_{v_i^c \epsilon V^c} a_{ij}\, \alpha_{i,t} \leq\ \beta_{j,t} x_j,\ \forall\ v_j^v \in\ V^v, t \epsilon \{1,2\} \tag{4.5}$$

$$\sum_{v_j^v \epsilon V^v} a_{ij}\ \beta_{j,t} \geq \alpha_{i,t}\ , \forall\ v_i^c \in\ V^c, t \epsilon \{1,2\} \tag{4.6}$$

Constraint (4.5) ensures that the sum of the requirements of application components allocated to a virtual machine cannot exceed its capacity. Constraint (4-5) also guarantees that $x_j = 1$ if $\sum_{v_i^c \epsilon V^c} a_{ij} > 0$ e.g. if there is a mapping to the virtual node v_i^c and 0 otherwise.

Constraint (4.6) states that each application component gets at least its computing requirement.

Constraints to ensure that $z_{(j,j')}^{(i,i')} = a_{ij}.\, a_{i\prime j\prime}$ are as follows:

$$\sum_{v_j^v \in V^v} z_{(j,j')}^{(i,i')} = a_{i\prime j\prime}, \forall\ v_i^c, v_{i\prime}^c\ \in\ V^c, \forall\ v_{j\prime}^v \in\ V^v \tag{4.7}$$

$$\sum_{v_{j\prime}^v \in V^v} z_{(j,j')}^{(i,i')} = a_{ij}, \forall\ v_i^c, v_{i\prime}^c\ \in\ V^c, \forall\ v_j^v \in\ V^v \tag{4.8}$$

$$a_{ij} + a_{i\prime j\prime} -\ z_{(j,j')}^{(i,i')} \leq 1, \forall\ v_i^c, v_{i\prime}^c\ \in\ V^c, \forall\ v_j^v, v_{j\prime}^v \in\ V^v \tag{4.9}$$

Constraints (4.7) and (4.8) ensure the correlation between a and z. Constraint (4.9) ensure the coherence between application nodes mappings and their associated dependency links mappings.

We use the Multi-Commodity Flow problem (MCF) for the link mapping which maximizes the link utilization while preferring paths with minimal costs such that:

- *Capacity constraints*

$$\sum_{e^v_{(j_1,j'_1)} \in E^v} b^{(j,j')}_{(j_1,j'_1)} + \sum_{e^v_{(j'_1,j_1)} \in E^v} b^{(j,j')}_{(j'_1,j_1)} \leq \gamma_{(j,j')} y_{(j,j')} \quad \forall\, e^v_{(j,j')} \in E^v \tag{4.10}$$

Constraint (4.10) ensures the network capacity constraint. Constraint (4.10) also guarantees that $y_{(j,j')} = 1$ if $\sum_{e^v_{(j_1,j'_1)} \in E^v} b^{(j,j')}_{(j_1,j'_1)} + \sum_{e^v_{(j1',j1)} \in E^v} b^{(j,j')}_{(j'_1,j_1)} > 0$ e.g. if there is a mapping to the virtual link $e^v_{(j,j')}$ and 0 otherwise.

- *Flow conservation constraints*

$$\sum_{e^v_{(j,l)} \in V^v} b^{(j,l)}_{(j_1,j'_1)} - \sum_{e^v_{(l,j')} \in V^v} b^{(l,j')}_{(j'_1,j_1)} = 0 \;\forall\, v^v_{j_1}, v^v_{j_1'} \in V^v, \quad \forall\, v^v_l \in V^v / \{v^v_{j1}, v^v_{j1'}\} \tag{4.11}$$

Constraint (4.11) ensures edge continuity. In fact, the sum of the incoming flow must be equal to the sum of the outgoing flow.

- *Required flow constraint at the source*

$$\sum_{e^v_{(j,h)} \in V^v} b^{(j,h)}_{(j,j')} - \sum_{e^v_{(h,j)} \in V^v} b^{(h,j)}_{(j,j')} = bw(j,j') \quad \forall\, v^v_j, v^v_{j'} \in V^v \tag{4.12}$$

Constraint (4.12) ensures the flow conservation at the source. It incurs that a flow must exit its source node completely.

- *Required flow constraint at the destination*

$$\sum_{v_h^v \in V^v} b_{(j,j')}^{(h,j')} - \sum_{v_h^v \in V^v} b_{(j,j')}^{(j',h)} = bw(j,j') \; \forall \; v_j^v, v_{j'}^v \in \; V^v \tag{4.13}$$

Constraint (4.13) ensures the flow conservation at the destination. It incurs that a flow must enter its terminating node completely.

Accordingly, the objective function is given by:

$$Min\left(\sum_{v_j^v \in V^v} x_j \eta(v_j^v) + \sum_{e_{(j,j')}^v \in E^v} y_{(j,j')} \eta(e_{(j,j')}^v)\right) \tag{4.14}$$

S. T.

$$(4.3), (4.4), (4.5), (4.6), (4.7), (4.8), (4.9),$$
$$(4.10), (4.11), (4.12), (4.13)$$

$$x_j = \begin{cases} 0 & if \sum_{e_{(j,j')}^v} y_{(j,j')} - \sum_{e_{(j',j)}^v} y_{(j',j)} = 0 \\ 1 & otherwise \end{cases} \tag{4.15}$$

$$a_{ij} \in \{0,1\}, b_{(j_1,j_1')}^{(j,j')} > 0, x_j \in \{0,1\}, y_{(j,j')} \in \{0,1\} \tag{4.16}$$

The first part of the objective function aims to minimize the rental cost of mapped virtual machines and the second part of the objective function aims to minimize the overall rental cost of mapped network edges.

(4.15) defines the correlation x_j and $y_{(j,j\prime)}$. It indicates that a virtual node v_j^v is allocated if the incoming flow to that node is not equal to the outcoming flow. Consider θ a big number. (4.15) can be linearized as follows:

$$\begin{aligned}
&\sum_{e^v_{(j,j\prime)}} y_{(j,j\prime)} - \sum_{e^v_{(j\prime,j)}} y_{(j',j)} \leq \theta\, x_j \\
&\sum_{e^v_{(j\prime,j)}} y_{(j\prime,j)} - \sum_{e^v_{(j,j\prime)}} y_{(j,j\prime)} \leq \theta\, x_j \\
&\sum_{e^v_{(j,j\prime)}} y_{(j,j\prime)} - \sum_{e^v_{(j\prime,j)}} y_{(j',j)} \geq x_j \\
&\sum_{e^v_{(j\prime,j)}} y_{(j\prime,j)} - \sum_{e^v_{(j,j\prime)}} y_{(j,j\prime)} \geq x_j
\end{aligned} \tag{4.17}$$

Constraint (4.16) indicates real domain of the variable $b^{(j,j')}_{(j_1,j_1')}$ and binary domains of variables a_{ij} , x_j and $y_{(j,j\prime)}$.

4.6 OptiDep algorithm

This section is dedicated to addressing the objective O3 which is about developing an algorithm according to the optimization model to optimally map application components to available resources.

Mapping application graphs into shared cloud infrastructure networks expressed in our optimization model as a Mixed Integer Linear Programming problem is known to be an NP-hard problem. Therefore, we propose an algorithm, named OptiDep that is solved using GLPK, an LP solver.

The proposed OptiDep algorithm is initiated by a request e.g. application to be deployed. The available cloud resources are calculated and the infrastructure graph is built. After that, the matching process outputs the virtual graph G^v. Next, OptiDep analyzes the input of the

application and designs the logical graph of the application to be deployed. If there is a locally constrained component, it updates its component's compute capacities to null and add a virtual node with null compute capacities to be mapped to it. The *minimal_cost_assignment* function is then called. The function takes as input the application graph G^c and the virtual graphs G^v and rental costs η. In case of a successful mapping, the algorithm returns the mapped virtual resources $\{ [a_{ij}]_{v_i^c \in V^c, v_j^v \in V^v}, [b^{(j,j')}_{(j_1,j_1')}]_{e^v_{(j,j')} \in E^v, v^v_{j_1}, v^v_{j_1'} \in V^v} \}$ with the computed minimum cost F_{min} .Otherwise, the request is rejected.

Algorithm 4.2 OptiDep

OptiDep

Input: virtual graph $G^v = (V^v, E^v)$, application graph $G^c = (V^c, E^c)$, Cost η

Output: $Mapping = \{ [a_{ij}]_{v_i^c \in V^c, v_j^v \in V^v}, [b^{(j,j')}_{(j_1,j_1')}]_{e^v_{(j,j')} \in E^v, v^v_{j_1}, v^v_{j_1'} \in V^v} \}$

1. $r \leftarrow 0$;
2. **for all** $v_i^c \in V^c$
3. **if** $(islocale(v_i^c) == true)$) then
4. set $\alpha_{i,1} \leftarrow 0$;
5. set $\alpha_{i,2} \leftarrow 0$;
6. $s \leftarrow 0$;
7. Add node v^v_{J+s} to the virtual graph G^v with $\beta_{J+s,1} = 0, \beta_{J+s,2} = 0$;
8. $s \leftarrow s + 1$;
9. **end if**
10. **end for**
11. $r \leftarrow$Solve *Minimal_cost_assignment*(G^c, G^v, η)
12. **if** $r \neq 0$ **then**
12. reject G^c;
13. break;
14. **else**
15. **return** $\{ [a_{ij}]_{v_i^c \in V^c, v_j^v \in V^v}, [b^{(j,j')}_{(j_1 j_1')}]_{e^v_{(j,j')} \in E^v, v^v_{j_1}, v^v_{j_1'} \in V^v} \}$ and the optimal cost F_{min}
16. **end if**

Algorithm 4.2 OptiDep (continued)

Function *Minimal_cost_assignment*(G^c, G^v, η)

1. $Min\ \left(\sum_{v_j^v\ \epsilon\ V^v} x_j \eta(\ v_j^v) + \sum_{e^v_{(j,j')}\ \epsilon\ E^v} y_{(j,j')} \eta(\ e^v_{(j,j')})\right)$
2. $\sum_{v_j^v \epsilon V^v} a_{ij} = 1\ \forall\ v_i^c\ \epsilon\ V^c$
3. $\sum_{v_i^c \epsilon V^c} a_{ij}\ \alpha_i \leq\ \beta_{j,t} x_j,\ \forall\ v_j^v\ \epsilon\ V^v, t\ \epsilon \{1,2\}$
4. $\sum_{v_j^v \epsilon V^v} a_{ij}\ \beta_{j,t} \geq \alpha_{i,t}\ , \forall\ v_i^c\ \epsilon\ V^c, t\ \epsilon \{1,2\}$
5. $\sum_{e^c_{(i,i')} \epsilon E^c} \delta_{(i,i')}\ z^{(i,i')}_{(j,j')} = bw(j,j')\ \forall\ v_j^v, v_{j'}^v\ \epsilon\ V^v$
6. $\sum_{v_j^v \in V^v} z^{(i,i')}_{(j,j')} = a_{i'j'}, \forall\ v_i^c, v_{i'}^c\ \epsilon\ V^c, \forall\ v_j^v\ \epsilon\ V^v$
7. $\sum_{v_{j'}^v \in V^v} z^{(i,i')}_{(j,j')} = a_{ij}, \forall\ v_i^c, v_{i'}^c\ \epsilon\ V^c, \forall\ v_j^v\ \epsilon\ V^v$
8. $a_{ij} + a_{i'j'} -\ z^{(i,i')}_{(j,j')} \leq 1, \forall\ v_i^c, v_{i'}^c\ \epsilon\ V^c, \forall\ v_j^v, v_{j'}^v \epsilon\ V^v$
9. Consider Multicommodity flow problem (MCF) constraints from (4-9), (4-10), (4-11),(4-12)
10. Consider correlation constraint between x_j and $y_{(j,j')}$ from (4-17)
11. **If** *Successful mapping* then
12. | **return** 0;
13. **else**
14. | **return** 1;
15. **end**

4.7 Proposed architecture

As mentioned in our objective O4, we need to design an architecture to automate the resource provisioning and application deployment process. To that end, we have built an application virtualization platform.

The platform is composed of a software architecture which has: 1) a decision module; and 2) a deployment module. Figure 4.3 provides a high-level view of the platform architecture with

a description of each module including its architectural elements and how they interact with each other.

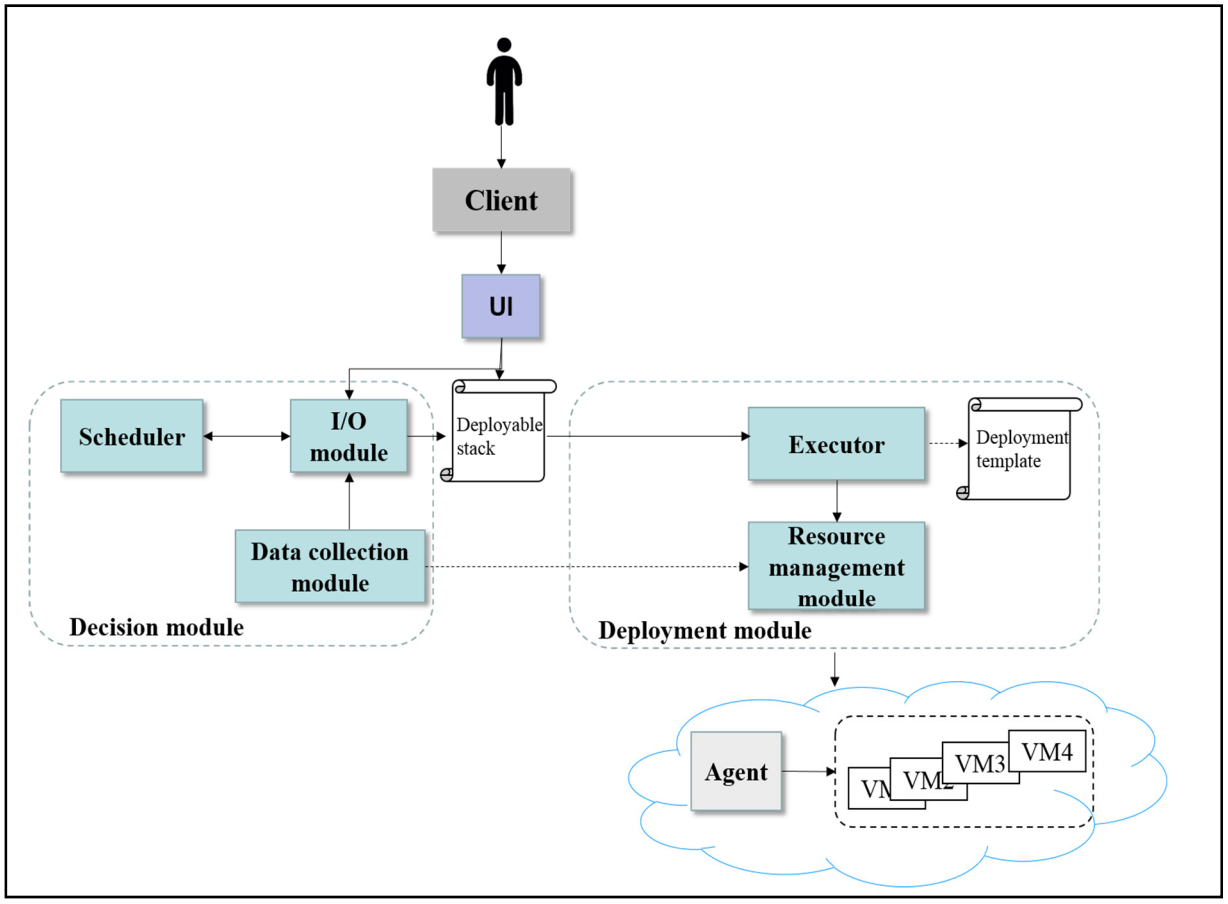

Figure 4.3 Application virtualization system

4.7.1 Decision module

The decision module is responsible for scheduling, upon request, the applications to be deployed in the cloud environment. As we can see in Figure 4.3, the decision module is composed of a scheduler, an I/O module and a data collection module.

- *Scheduler*

  The scheduler is responsible for calculating the optimal placement for the complex services to be deployed;

- *I/O module*

  The I/O module sends data on available cloud resources, like CPU, memory, and bandwidth, to the scheduler. The module also provides the scheduler with the applications information specified by end users by communicating with the user interface (UI).

  After performing the mapping process, the scheduler sends the result to the I/O module. The I/O module creates a deployable stack containing the result along with other information specified by the end-user e.g. software modules, protocols, and sends it to the deployment module;

- *Data collection module*

  The data collection module is responsible for collecting the available cloud resources, like the available CPU, memory, and bandwidth capacity.

The decision process, presented in Figure 4.4, is triggered by a deployment request. It calculates the available cloud resources and builds accordingly the cloud infrastructure graph. It also analyzes the application requirements, updates the list of application components and designs logical graphs of applications to be deployed. It calls OptiDep to build an optimal deployment plan. If the problem is unsolvable, the process rejects the request.

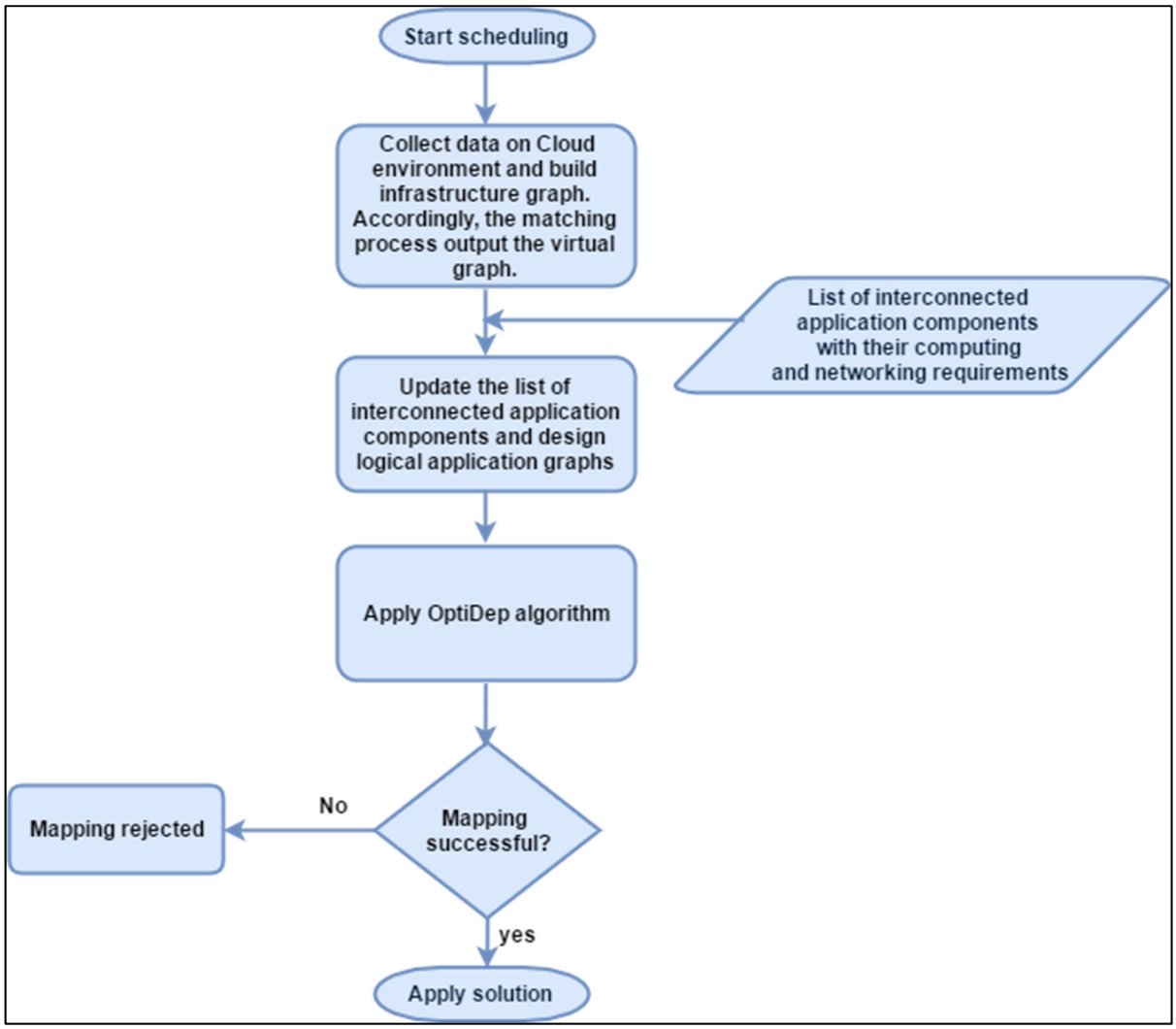

Figure 4.4 Scheduling Flowchart

4.7.2 Deployment module

The main responsibility of the deployment module is to automatically deploy applications. This module receives a deployable stack containing the result of the decision module and a set of other parameters specified by the user e.g. software modules to be installed. It allocates compute and network resources and deploys application components.

This module has been integrated into an existing cloud management system, namely OpenStack(Sefraoui, Aissaoui et Eleuldj, 2012).

4.7.2.1 Architecture

The deployment module as shown in Figure 4.3 is composed of:

- *Deployable stack:* is the output of the decision module. It contains the set of parameters required to deploy an application such as the number and types of VM instances, specific constraints, network configuration, etc. It also contains the application components to be deployed, e.g. databases, specific products, middleware, etc;
- *Executor:* is a service responsible for managing the deployment of applications. It provisions compute and network resources via the resource management module, monitors the state of the deployment, and acts as an information broker to help application configuration. The executor stores information about its deployments in a database;
- *Resource management module:* is responsible for the management of the full life cycle of virtual machines instances. This includes the allocation of a VM‘s disk, assignment of dynamic IP addresses to VM, allocating an image to the VM instance or providing a key to access the different services;
- *Agent:* collects information about the state of VM instances to make sure they have been successfully created.

4.7.2.2 Deployment module process

The deployment process is presented in Figure 4.5. It starts by a step that creates a deployment template for the application. End users may demand complex distributed services. Therefore, the deployment template is used to support multiple service instances. The deployment

template contains the different resources needed and ready to be built. It is run to order resources.

The system resolves the template parameters to ensure there is no error. Then, it passes the request to the resource management module to book resources and deploy application components. An application deployment model is then generated containing all deployment configurations for each application component. The system then verifies if any errors occurred during the creation of instances. If there is any, the system automatically retries the process.

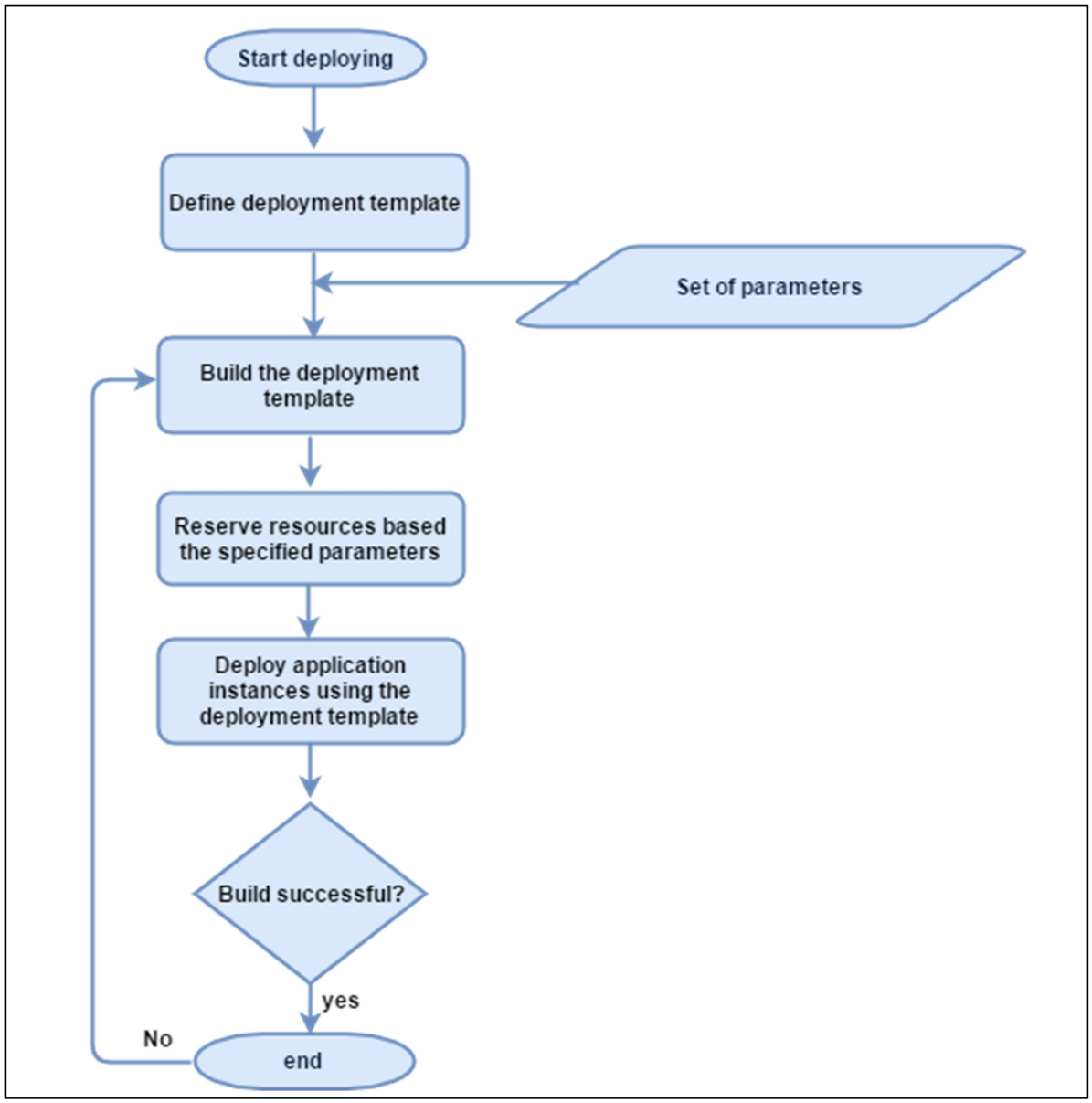

Figure 4.5 Deployment process flowchart

Conclusion

This chapter presented the research methodology. First, the system modeling is presented. Then, a Mixed Integer Linear Programming (MILP) model has been proposed to map home automation applications to the available cloud resources at minimal costs while maintaining the required QoS. Then, we have designed the OptiDep algorithm that will be used to solve the optimization model. Finally, we have proposed an application virtualization platform designed to automate the deployment of applications onto cloud environment. The proposed platform uses the mapping algorithm to calculate the optimal provisioning plan and then allocates resources accordingly to deploy applications.

CHAPTER 5

SYSTEM IMPLEMENTATION AND EVALUATION RESULTS

In this chapter, we first present the implementation of the proposed application virtualization platform. Then, we describe a case study of modeling dependencies between application requirements and QoS classes. The final section is dedicated to the evaluation results of the proposed application placement algorithm.

5.1 System implementation

5.1.1 Decision module implementation

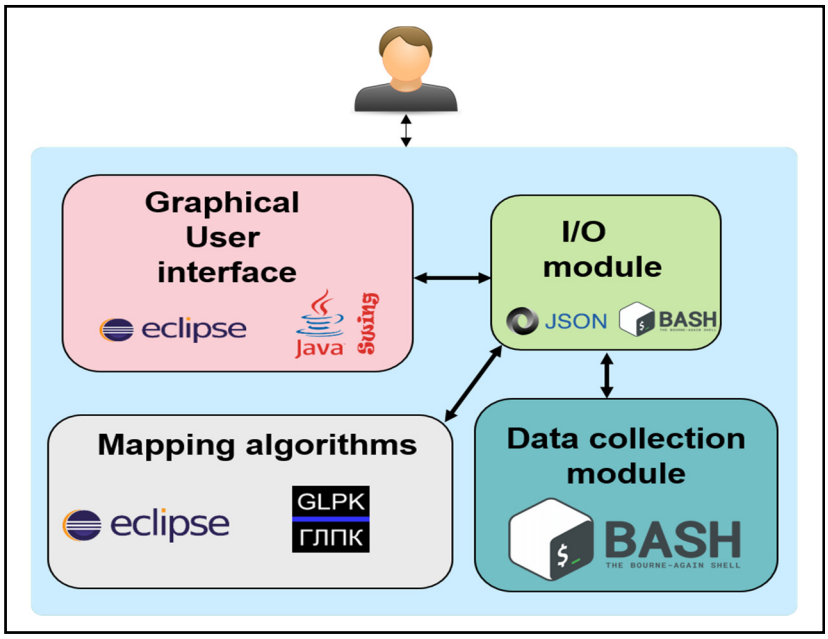

Figure 5.1 The implementation architecture of the decision module

The decision module is designed to perform the mapping of applications in a real cloud environment depending on the user requests.

It communicates with OpenStack to collect the available cloud resources through the data collection module. The decision module architecture is composed of four block modules as in Figure 5.1.

5.1.1.1 The I/O module

This module is responsible for information exchange. It communicates with the graphical user interface (GUI) to collect data as defined by the end user e.g. number of applications, computing capacities of application components in terms of CPU, memory, networking capacities in terms of bandwidth. The I/O module also interacts with the data collection module to get the available cloud resources. The I/O module is also responsible for launching the mapping algorithm, collecting the results of the mapping algorithms and putting them in a deployable stack. This module is implemented as a collection of Shell scripts and JSON files.

5.1.1.2 Graphical user interface

The graphical user interface (GUI) is an interface provided to the end user. It receives user specifications. The interface is implemented in Java using the Java Swing library.

5.1.1.3 Mapping algorithm

The mapping algorithm is used to resolve the application placement problem. The mapping algorithm is implemented as an Eclipse plug-in project using the GNU Linear Programming Kit (GLPK) solver.

5.1.1.4 Data collection module

The Data collection module communicates with the resource management module in order to collect the available compute and network capacity of cloud resources.

To summarize, the user specifies the applications to deploy with their computing and their networking requirements along with the location and the communication constraints. The I/O module collects this data and put it in an input file. It triggers the scheduling algorithm to compute an application placement plan. Finally, the I/O module updates the deployment stack with the results from the mapping algorithm.

5.1.2 Deployment module implementation

5.1.2.1 Overview

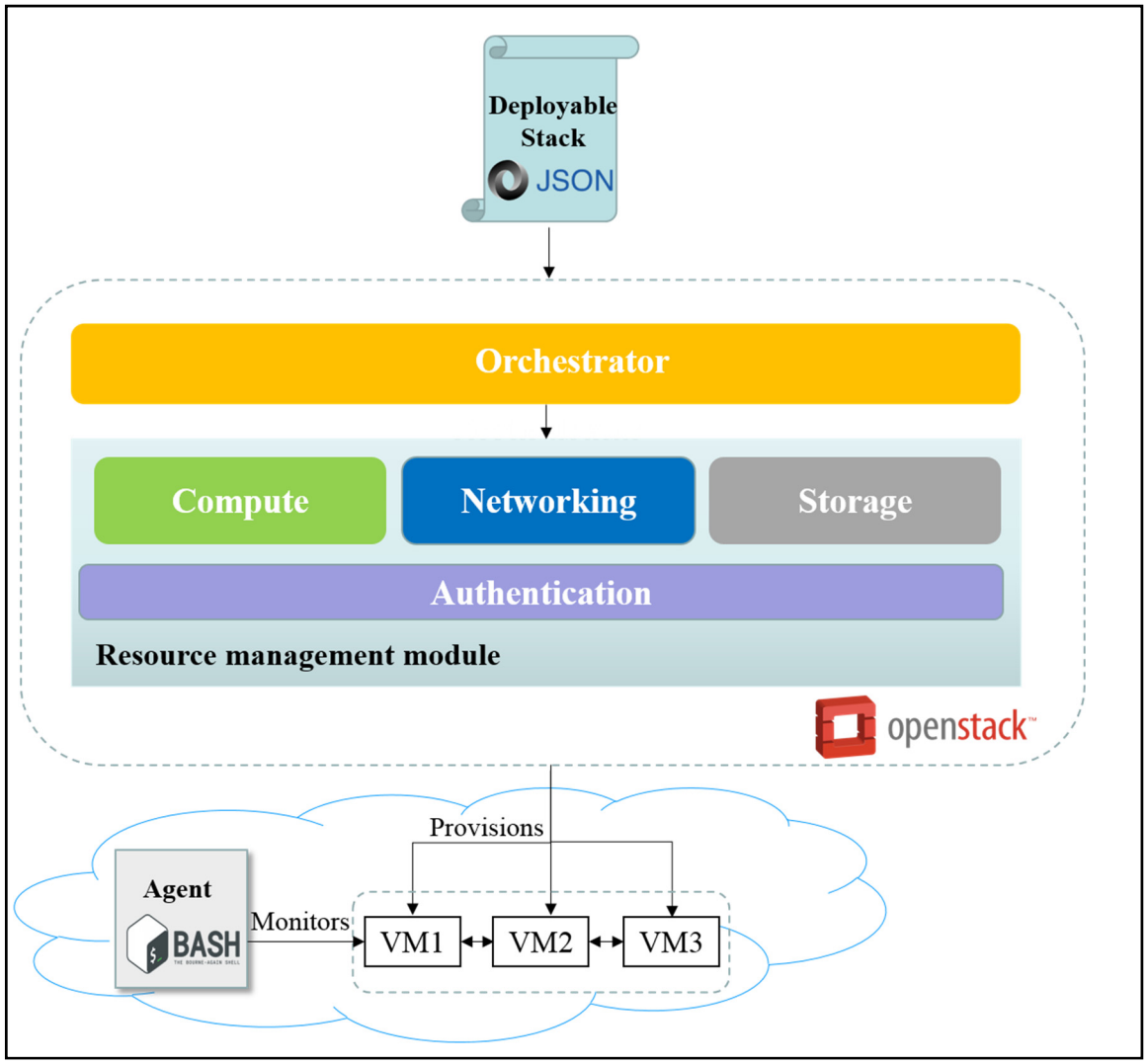

Figure 5.2 Deployment module implementation architecture

The deployment module is designed to automatically deploy complex distributed applications onto the cloud. The deployment module architecture includes the following components (Figure 5.2):

- *Deployable stack*

  It is a JSON formatted file that includes the requirements (capacity and non-capacity e.g. availability, location, processor type, QoS parameters, etc) of each application component. It is invoked by the decision module to configure the placement of the application component and the graphical user interface (GUI) to enter the other non-capacity requirements;
- *Resource management module*

  The module manages the deployment of applications. It ensures the provisioning of cloud resources. It is operated by OpenStack services (details in the next section);
- *Agent*

  The module is a Shell script responsible for collecting information about the created VM instances, reporting the state of the running VMs. It checks whether the application component is deployed successfully or not and monitor the instances in case of failures;

In order to implement the deployment module, we have set up a Cloud Testbed on OpenStack.

5.1.2.2 OpenStack

OpenStack is a free and open-source software platform for cloud computing, which is deployed as an infrastructure-as-a-service (IaaS) to provide a private cloud. The software platform consists of interrelated components that control diverse, multi-vendor hardware pools of processing, storage, and networking resources throughout a data center. The main components of OpenStack are as followed:

- *OpenStack Compute:* is responsible for creating and managing instances using the provided images by the service glance;
- *OpenStack Keystone:* provides authentication service to access the different other OpenStack services;
- *OpenStack Networking:* is in charge of networking management. It is responsible for managing the IP addresses, VLANs, and firewalls for the created instances;
- *OpenStack Glance:* is responsible for providing disk and server images when creating virtual machine instances;
- *OpenStack Heat:* This component acts as an orchestrator that manages multiple Cloud applications through REST APIs. Heat allows users to describe deployments of complex cloud applications in text files called templates. These templates are then parsed and executed by the Heat engine;
- *OpenStack Cinder:* This component called also block storage is responsible for providing volumes to running instances.

OpenStack provides a number of projects. Each project or also called as a tenant is a logical grouping of users where each user consumes cloud resources. A project has a defined quota in terms of resources e.g. RAM, IP addresses, number of cores, size of storage volumes… within the cloud environment and totally isolated from other projects. This quota is defined depending on the project and the contractual agreement. Each tenant can allocate a public IP address called a floating IP and attach it to a VM instance. This public IP address is the only way to connect to this VM instance from outside.

5.1.2.3 Testbed implementation

We have set up a cloud Testbed which is composed of two sites; the first site is located in École de Technologie Supérieure in Montréal, the second site is located in Moncton.

The architecture of the Cloud Testbed is composed of six nodes; five nodes on the first site and one node on the second site. At the Montreal site, the Controller, Neutron, and Storage nodes

are deployed as virtual machines on two servers. The first server hosts the controller and the neutron node, the second server hosts the storage node. Two compute nodes are deployed, each on a separate server. At the Moncton site, there is a single compute node deployed on a dedicated server. Each virtual machine hosting the controller, the neutron, and the storage has 6 CPU, 12 GB of RAM and 20 GB of storage. The two physical servers at Montreal site have each one 12 CPU, 24 GB of RAM and 251 GB of storage. The server at Moncton has 16 CPU, 63 GB of RAM and 300 GB of storage.

A high-level view of the Testbed is presented in Figure 5.3.

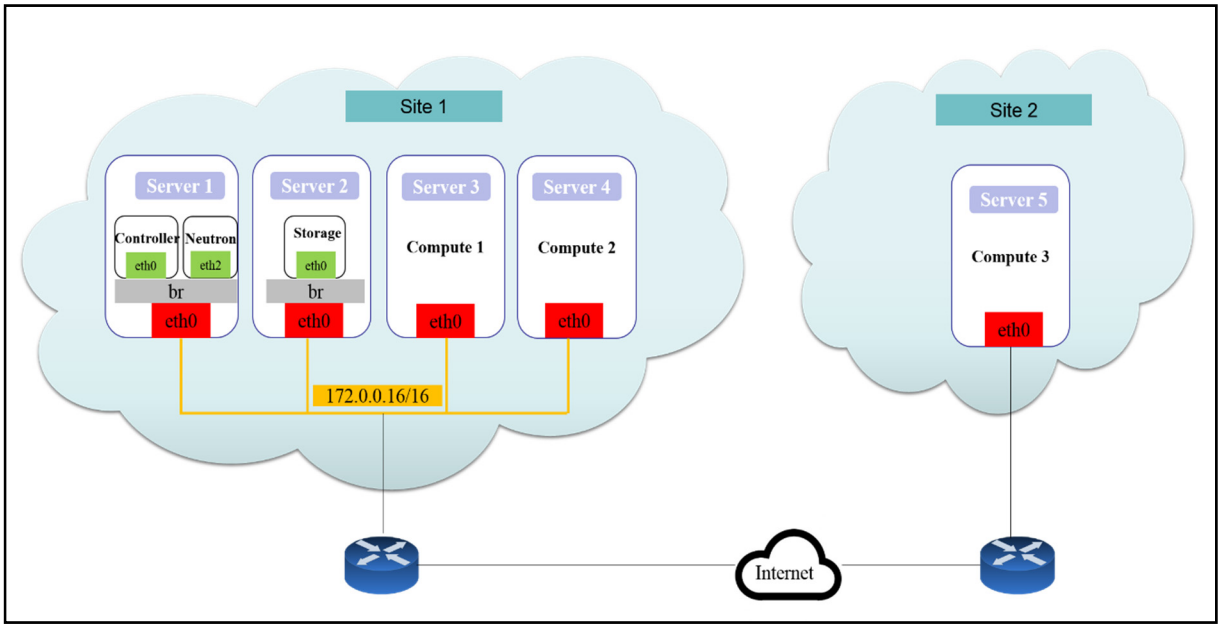

Figure 5.3 Cloud Testbed

5.1.2.4 Pricing model

Our pricing model is based on Amazon Web Services (AWS) (Cloud, 2011) pricing scheme that charges its customers depending on the location and per hour. We set up three services as seen in Table 5.1 with different server locations different characteristics (e.g. availability, computing resources). We assume that incoming traffic is not charged.

Table 5.1 Pricing model

Service	Server Names	Prices
	Ets-blade-7	Tiny: 0.02 $/h Small:0.04 $/h Medium:0.07 $/h Large: 0.25 $/h Xlarge: 0.5 $/h Bandwidth: 0.1 $/Go
2	Ets-blade-19	Tiny: 0.01 $/h Small: 0.02 $/h Medium: 0.05 $/h Large: 0.16 $/h Xlarge: 0.4 $/h Bandwidth: 0.08 $/Go
3	SEPIA	Tiny: 0.03 $/h Small: 0.05 $/h Medium: 0.08 $/h Large: 0.3 $/h Xlarge: 0.55 $/h Bandwidth: 0.15 $/Go

There are five types of cloud instances as indicated in Table 5.2:

Table 5.2 VM instances characteristics

Instance type	CPU	RAM (GB)	Disk (GB)
tiny	1	0.5	1
small	1	2	2
medium	2	4	5
large	4	8	10
Xlarge	8	16	20

5.1.2.5 Example of a complex service deployment

Consider the example of video monitoring application. First, a camera set on the front door of a house captures images and video. Then, this video is transferred to the Cloud where at first videos are analyzed and whenever there is a motion detected, the video is saved and uploaded to a web server for later visualization and then, the user is notified. An overview of the capacity and non-capacity requirements is provided in Table 5.3.

Table 5.3 Application components' requirements

Application component & Dependencies	Capacity requirements	Non-capacity requirements
Video/image transferring (VT)	CPU: 2 RAM: 0.6 GB	Location: Local Protocol: HTTP
Motion detection (MD)	CPU:1 RAM: 1 GB	Location: Remote Protocol: HTTP
Video/Image saving (IS)	CPU: 2 RAM: 0.9 GB	Location: Remote Protocol: HTTP
Upload module (UM)	CPU: 3 RAM: 2 GB	Location: Remote Protocol: HTTP
User notification (UN)	CPU: 1 RAM: 0.5 GB	Location: Remote
Link “VT” → “MD”	Bandwidth: 5 GB/h	None
Link “MD” → “IS”	Bandwidth: 2 GB/h	None
Link “IS” → “UM”	Bandwidth: 2 GB/h	None
Link “UM” → “UN”	Bandwidth: 0.5 GB/h	None

An example of specifying requirement through the GUI is shown in Figure 5.4.

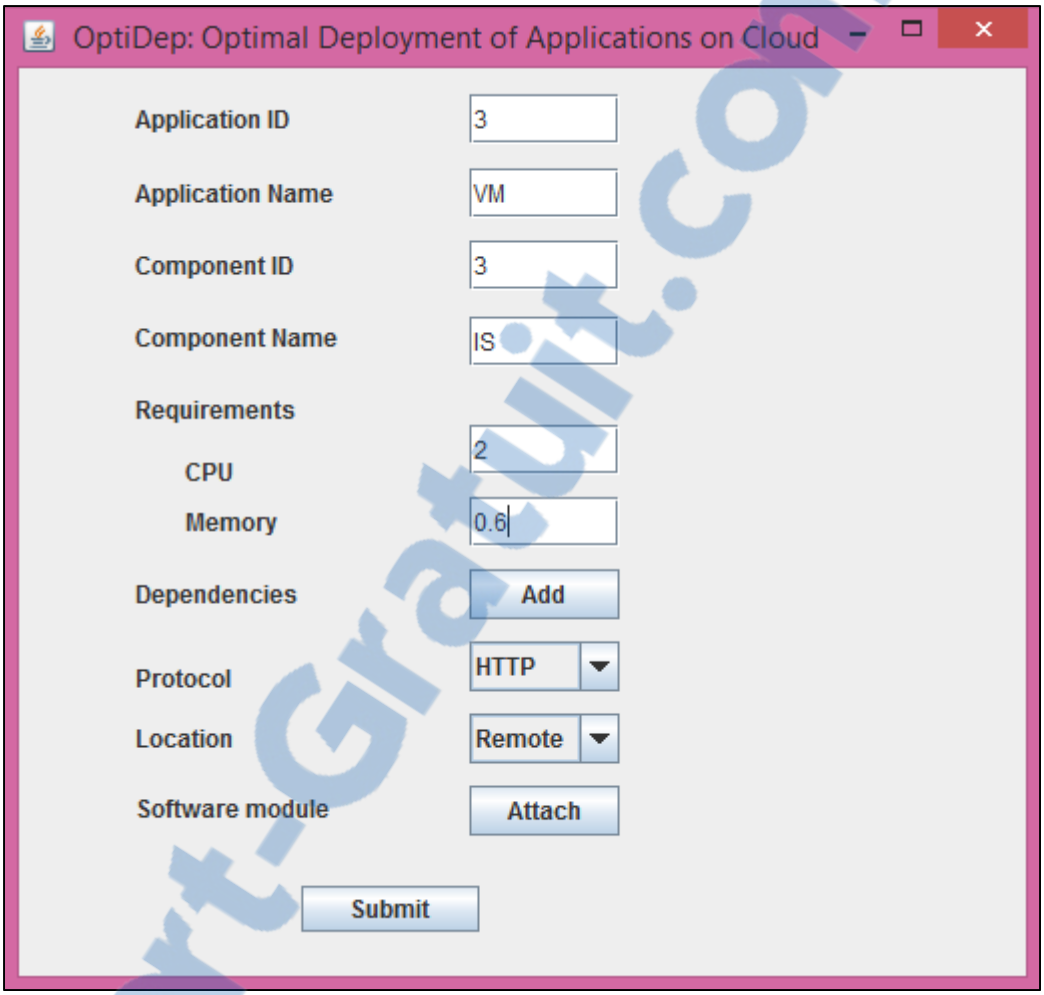

Figure 5.4 End user requirement specifications interface

The incoming request is analyzed by the decision module that will compute the mapping based on available Cloud resources and the application requirements, and save results in a deployable stack file as shown in Annex I. Table 5.4 summarizes the result of the mapping.

Table 5.4 Mapping results of application components

Application component	Instance flavor	Server
Motion detection (MD)	Small	Ets-blade-7
Video/Image saving (IS)	Medium	Ets-blade-19
Upload module (UM)	Large	Ets-blade-19
User notification (UN)	Small	Ets-blade-19

Next, we update the deployment template for each application component with parameters retrieved from the deployable stack e.g. type of flavor, server, etc. After that, we deploy applications using a master deployment template as shown in Annex II that defines the application and coordinates between application components and a deployment template for each application component as shown in Annex III.

5.2 Resources requirements model: Case study

Consider two components of the video monitoring application: the streamer and the motion detector. We model the dependency of these two components by assessing the compute and network requirements of the two components while varying QoS classes (in our case video resolution) and then applying statistical regression analysis.

5.2.1 Evaluation of compute and network requirements

In order to assess the network dependency between the two components, we have considered five types of video resolutions; very low, low, standard, HD and full HD respectively quantified as 1 to 5 where 1 corresponds to the very low resolution and 5 to the full HD resolution. Characteristics of the different video resolutions are as indicated in Table 5.5.

Table 5.5 Video resolution characteristics

Video resolution type	Horizontal resolution	Vertical resolution
Very low	352	240
Low	480	360
Standard	858	480
HD	1280	720
FullHD	1920	1080

5.2.1.1 Evaluation of the CPU requirements

- *Streaming service*

Figure 5.5 shows the behavior of CPU usage of the streaming service (ST) when varying the video resolution.

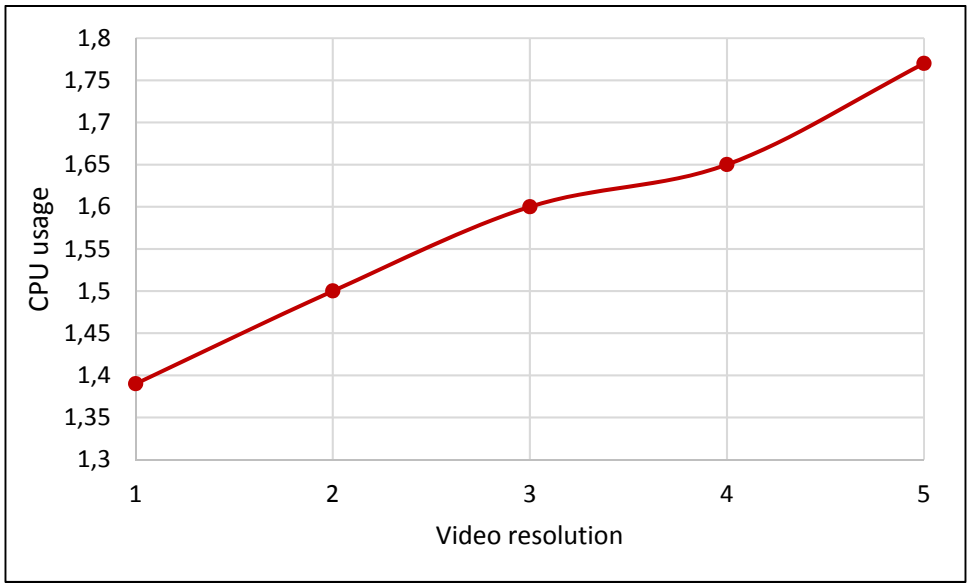

Figure 5.5 CPU usage versus of the ST service video resolution

- *Motion detection service*

Figure 5.6 depicts the evolution of the CPU usage of the motion detector (MD) when varying the video resolution from very low quality to full HD quality.

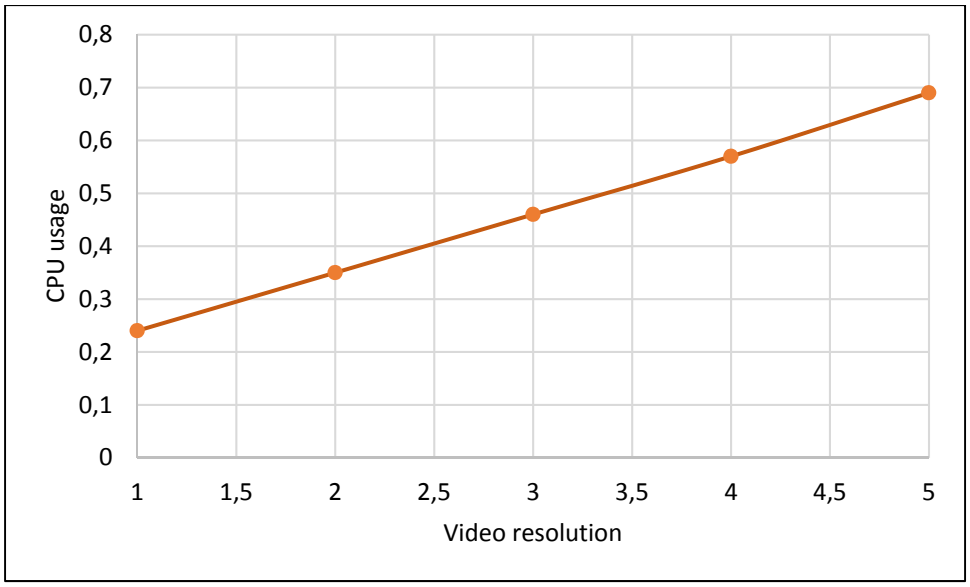

Figure 5.6 CPU usage of the MD service versus video resolution

5.2.1.2 Evaluation of memory requirements

- *Streaming service*

  Figure 5.7 shows the memory requirement of the Streaming service when varying the video resolution.

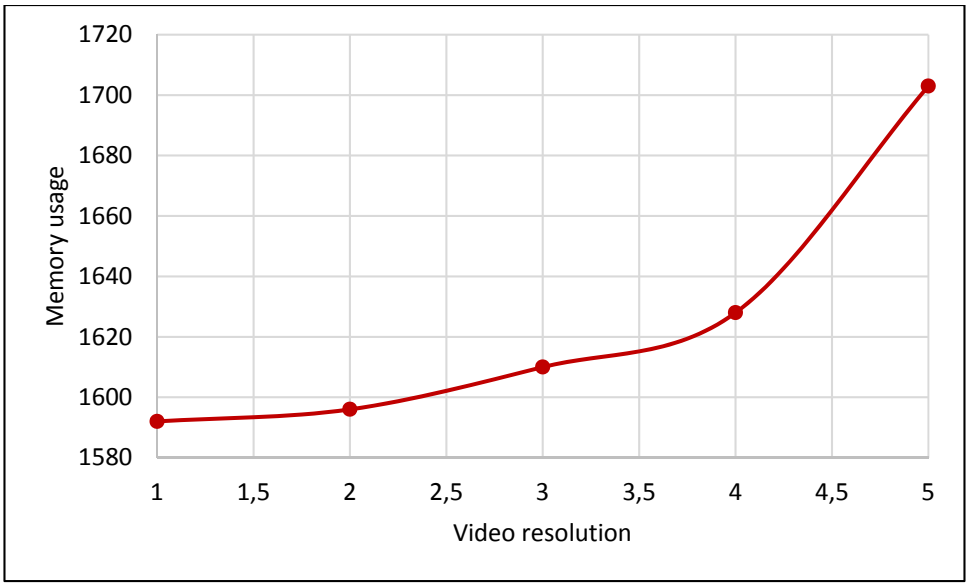

Figure 5.7 Memory usage of the ST service versus video resolution

- *Motion detection service*

  Figure 5.8 shows the assessment of the memory usage of the motion detection service when varying the video resolution.

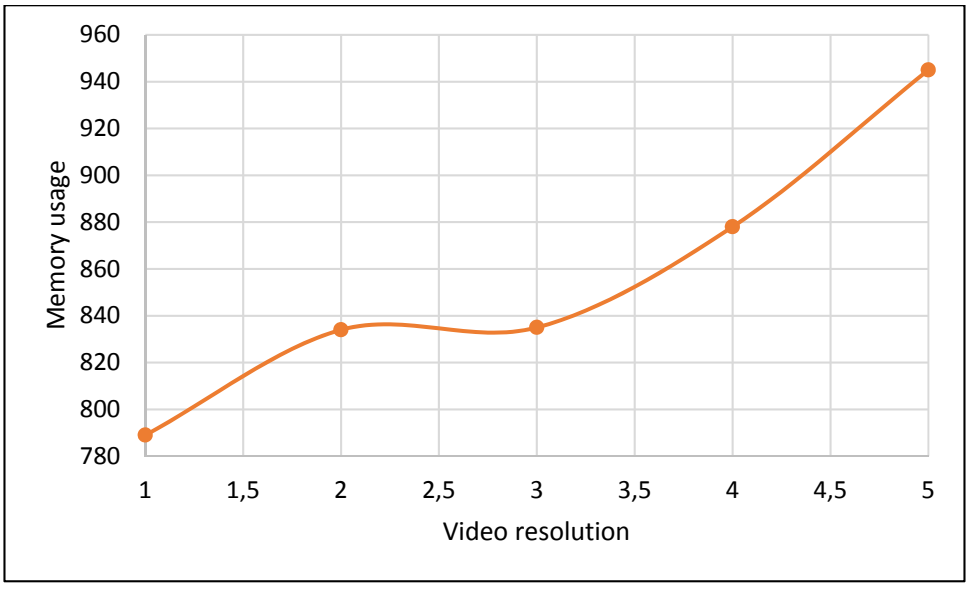

Figure 5.8 Memory usage of the MD service versus video resolution

5.2.1.3 Evaluation of bandwidth requirements

Evaluation of the bandwidth requirements between the motion detection (MD) service to the streaming (ST) service is indicated in Figure 5.9.

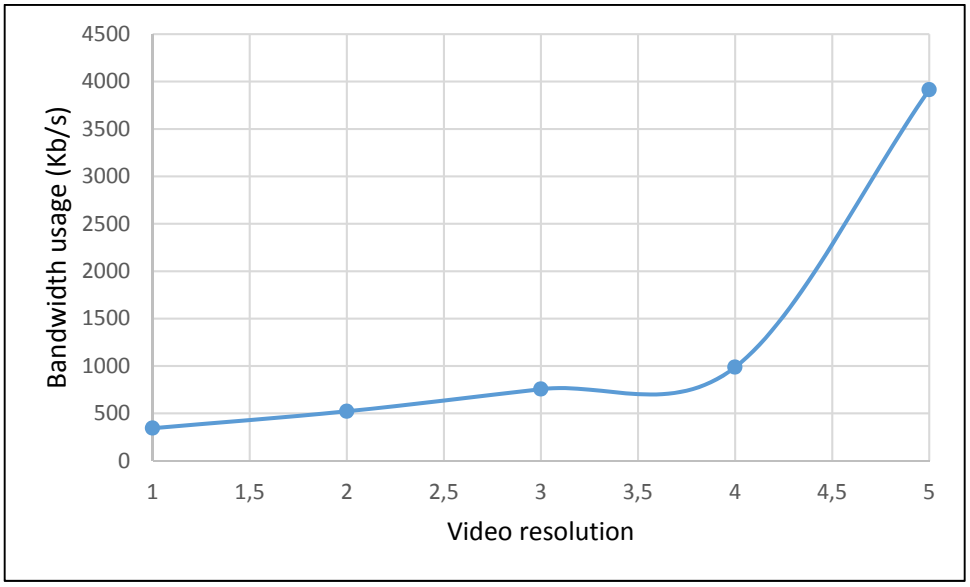

Figure 5.9 Bandwidth usage versus video resolution

5.2.2 Analytical results of application dependencies

After evaluating the requirements of the streaming and motion detection services in terms of CPU, memory, and bandwidth, we apply regression analysis to model the dependency between components requirements and QoS classes.

5.2.2.1 CPU

- *Streaming service*

  After applying statistical regression algorithms, we find out that CPU usage is linearly increasing according to video resolution. In fact, with $R^2 = 0,9872$ the model is :

$$\text{CPU} = 0{,}091 * \text{resolution} + 1{,}309 \quad (5.1)$$

- *Motion detection service*

  Similar to the streaming service, statistical regression analysis returns a linear correlation between the CPU usage of the motion detection service and video resolution with $R^2 = 0{,}9997$.

$$\text{CPU} = 0{,}112 * \text{resolution} + 0{,}126 \quad (5.2)$$

5.2.2.2 Memory

- *Streaming service*

  The correlation between memory usage of the streaming service and video resolution is modeled with $R^2 = 0{,}9672$ after calling the polynomial regression algorithm.

$$\text{Memory} = 10{,}429 * \text{resolution}^2 - 37{,}171 * \text{resolution} + 1622{,}6 \text{ (Kb)} \quad (5.3)$$

- *Motion detection service*

  Memory usage of the motion detection service is increasing exponentially when varying the video resolution with $R^2 = 0{,}9872$.

$$\text{Memory} = 755.2\, e^{0.04 * resolution} \text{ (Kb)} \quad (5.4)$$

5.2.2.3 Bandwidth

Statistical regression analysis algorithms with $R^2 = 0{,}885$ return the following correlation.

$$Bandwidth = 168{,}77 e^{0.55x\ Resolution} \text{(Kb/s)} \quad (5.5)$$

Regression analysis results demonstrate that bandwidth is increasing exponentially when varying the video resolution.

5.2.3 Discussion

In this part, the "Building application dependency models" algorithm has been applied to characterize dependencies between compute and network requirements and QoS classes for a video monitoring application. This method enables the use of well proven statistical regression analysis techniques in modeling application requirements dependencies as a step towards helping end users inputting their specifications. Results have shown that the proposed method can build the dependency model of an application with an average precision of 97%.

5.3 Evaluation results of the application placement algorithm

In order to evaluate the performance of OptiDep, we have compared our approach with Cost-VNE and Vineyard in terms of cost minimization, resource utilization, acceptance ratio and computation time.

5.3.1 Simulation environment

Table 5.6 Simulation parameters

Parameters	Values
Number of Applications nodes per request	[5,30]
Number of nodes in virtual graph	20
Connectivity of virtual nodes	0.5
Number of requests	3

We have implemented our solution in Java using the open-source linear programming toolkit GLPK to solve the mixed integer linear problem (MILP).

The virtual graph topology is composed of 20 nodes that are randomly generated. The nodes are connected with an average probability of 0.5 using a java tool that we have developed. We simulate five types of VMs: tiny, small, medium, large and xlarge instances. The prices of these VMs are as mentioned in Table 5.7 and the price of bandwidth is set to 0.08 $/GB per hour.

Table 5.7 The VMs

VM type	Price
Tiny	0.01 $/h
Small	0.02 $/h
Medium	0.05 $/h
Large	0.16 $/h
Xlarge	0.4 $/h

We increase the number of application nodes from 5 to 30, resulting in 26 scenarios. In our experiments, we have considered two different graph topologies: a sparse graph and a dense graph topology. A dense graph is a graph where the number of edges is close to the maximal O(n*(n-1)) n is the number of nodes and a sparse graph is a graph where the number of edges is close to the number of nodes O(n).

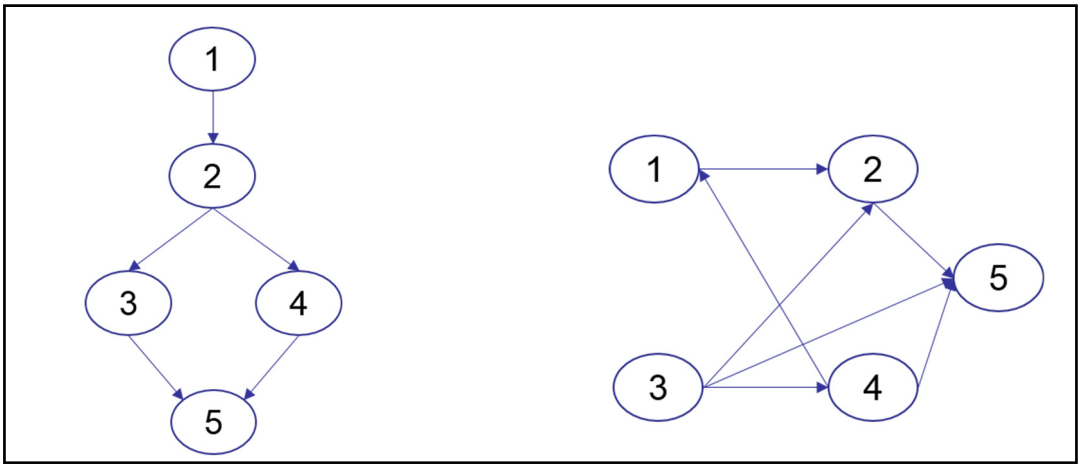

Figure 5.10 Example of a sparse graph (on the left) versus example a dense graph (on the right)

Each scenario has been repeatedly run 10 times.

A 2.4 GHz dual-processor PC with 8 GB of memory has been used for this experiment.

5.3.2 Experiment objectives

Though our solution returns the mapping with minimal costs, other metrics deserve our attention to better evaluate the efficiency and competitiveness of our approach. To summarize, the evaluation metrics are defined as follows:

5.3.2.1 Cost

The cost metric is calculated by summing up all the rental costs of VMs and networking links.

5.3.2.2 CPU utilization

The CPU utilization metric is measured by dividing the sum of demands in terms of CPU of all application components by the sum of mapped resources in terms of CPU.

5.3.2.3 Memory utilization

The memory utilization metric is measured by dividing the sum of demands in terms of memory of all application components by the sum of mapped resources in terms of memory.

5.3.2.4 Acceptance ratio

The acceptance ratio is the ratio of the number of successfully mapped application nodes and links divided by the overall number of application nodes and links.

5.3.2.5 Computation time

The computation time is the time needed for an algorithm to run. It is expressed in seconds.

5.3.3 Reference algorithms for comparison

In order to evaluate the performance of our approach, we have chosen CostVNE (Houidi et al., 2011) and Vineyard (Chowdhury, Rahman et Boutaba, 2012) as reference algorithms for comparison with OptiDep.

Cost-VNE is an exact virtual network embedding approach that minimizes the embedding cost in terms of allocated resources to the application requests. However, due to the fact that CostVNE model is no longer appropriate regarding current pricing models of cloud providers. Its equal resource utilization mapping will drastically differ in their rental costs. Moreover, CostVNE does not take into account smart home application-specific requirements.

The second approach is Vineyard. This algorithm offers a better coordination between the node mapping and the link mapping. It solves an MILP and multicommodity flow (MCF) problem through relaxation methods. It includes acceptance ratio, resource utilization and provisioning

cost in its formulation. The vineyard has proven to outperform other multiple mapping algorithms.

5.3.4 Evaluation method

We applied OptiDep, CostVNE and Vineyard algorithms to process clients' requests separately.
Each algorithm is applied to the same virtual graph and processes the same set of requests. We measured the mapping cost for each request, the resource utilization, the acceptance ratio and the execution time for each algorithm. We traced the evolution of results with the number of application nodes per request.

5.3.5 Evaluation results

5.3.5.1 Cost

Figures 5.11 and 5.12 show the rental costs of allocating cloud resources for each algorithm according to the number of application nodes. The values in the two figures represent the overall rental costs of mapping the same request graphs with the same number of nodes on the same virtual graph. We can see that OptiDep outperforms the two other approaches regardless of the density of the graph.

When the number of nodes is small, OptiDep and CostVNE tend to have almost the same performances but as we increase the number of nodes, we can see that from 15 nodes, the gap between the two approaches becomes large. A cost saving of 35% is obtained when the number of nodes is 30. This is because CostVNE tries to minimize the used resources to leave as much free capacity as possible regardless of resource costs, which will result in higher rental costs.

For the same number of nodes, the distance between the OptiDep and CostVNE approaches gets higher as the graph gets denser. In fact, our approach performs better when the number of requests increases.

We can also see that OptiDep outperforms Vineyard in both sparse and dense graphs. When the number of nodes is small, both algorithms tend to have close performances but from 10 nodes, the gap between the two approaches becomes significant. The overall cost for 30 nodes with OptiDep is 10.75 $/h whereas the overall cost with Vineyard is more than double (26$/h).

The Vineyard approach which consists of solving a linear problem by giving a rational value for each of the abstract nodes and abstract edges associated with a group of candidates of the substrate graph and then applying relaxation techniques deterministically or randomly (we choose the deterministic method) to choose one of the associated nodes to the abstract one as the best choice. This relaxation step is done in parallel for all the abstract nodes and edges resulting in a solution that does not take into account the whole topology and consequently, all possible solutions. The problem becomes worse as the graph gets denser and the number of application nodes increases, resulting in poor performances compared to the OptiDep approach. OptiDep is an exact approach that relaxes no constraint and provides a simultaneous node and link mappings, ensuring an optimal mapping solution. Moreover, Vineyard approach does not take into account the actual pricing model of cloud providers.

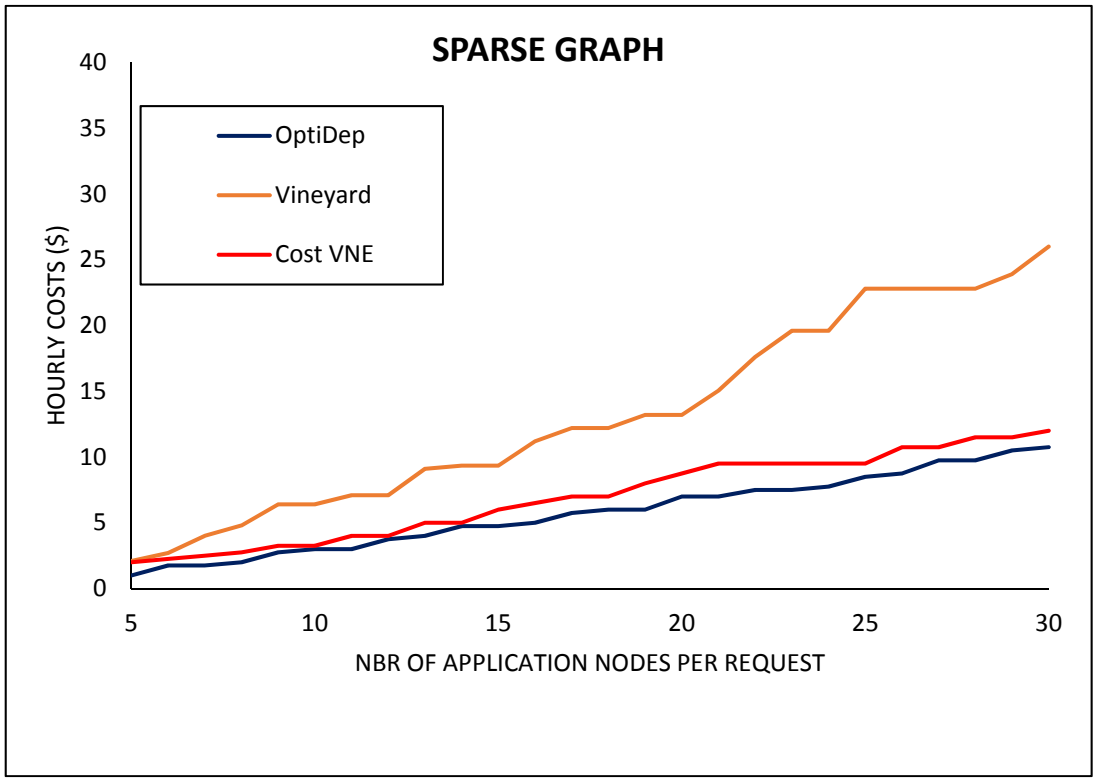

Figure 5.11 Hourly costs versus the number of application nodes per request in a sparse graph

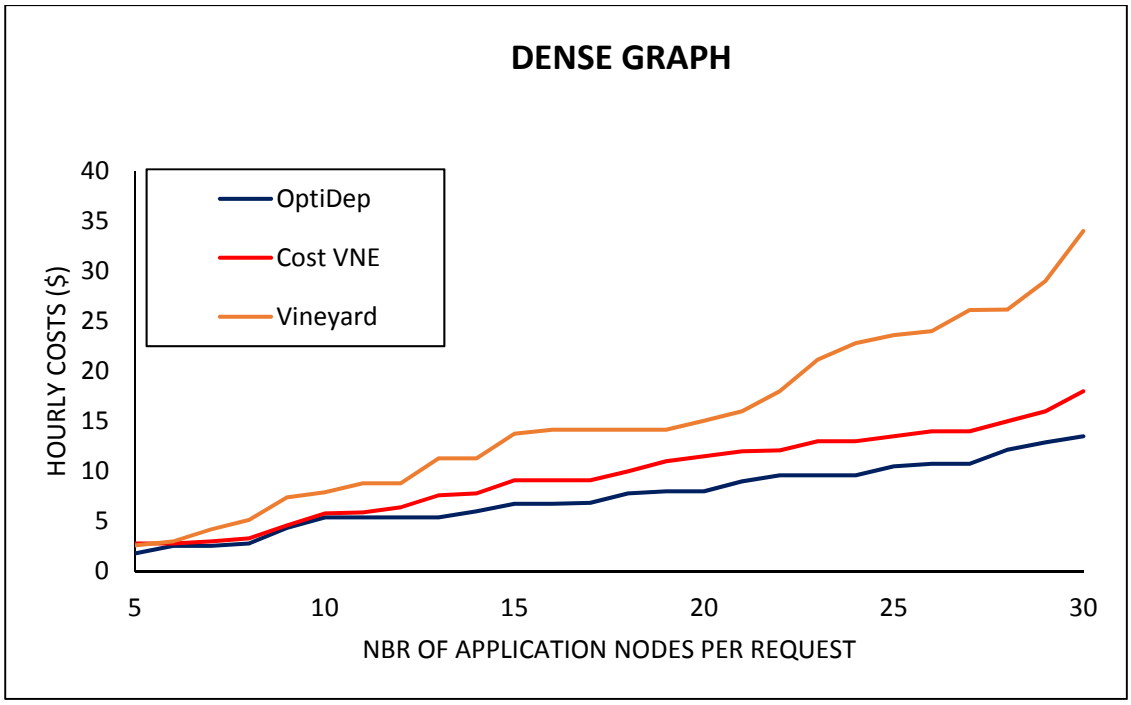

Figure 5.12 Hourly costs versus the number of application nodes per request in a dense graph

Figure 5.13 depicts the average cost of the three approaches after 26 scenarios respectively in a sparse and dense graph. We can conclude that our approach saves up to 20% in case of a sparse graph and up to 29 % in case of a dense graph compared to CostVNE.

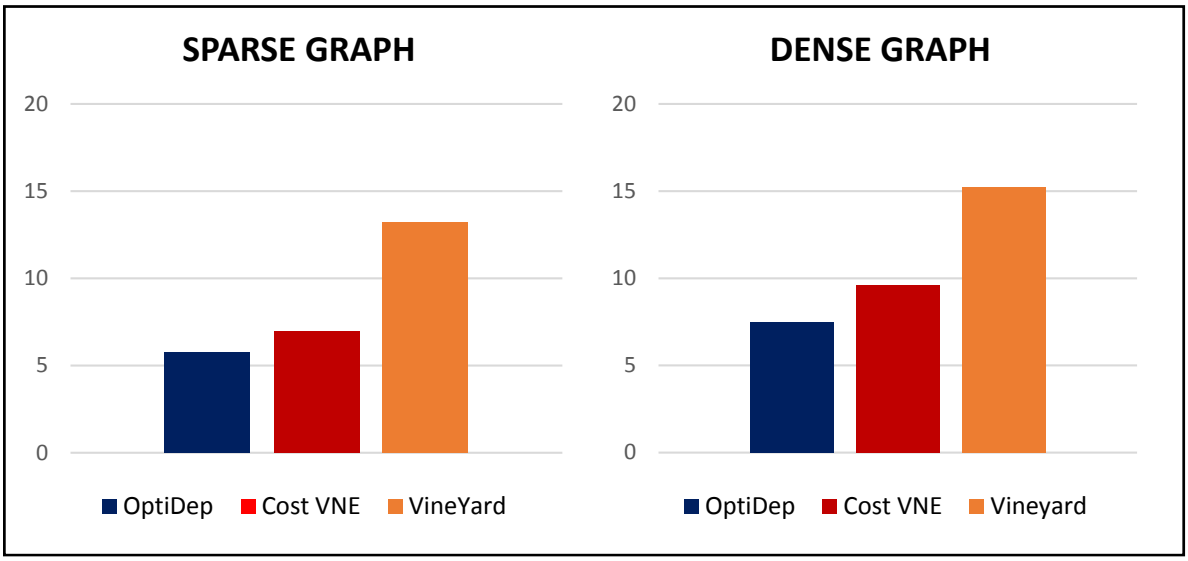

Figure 5.13 Average cost in case of a sparse graph versus the dense graph

5.3.5.2 Resource utilization

Figures 5.14, 5.15 and 5.16 depict the average resource utilization respectively in terms of CPU, memory, and bandwidth of the virtual graph by varying the number of application nodes per request.

- *CPU*

  We can see in Figure 5.14 that in average, CostVNE and OptiDep have the same behavior for the different number of nodes. This is due to the fact that CostVNE tries to minimize the allocated resource capacity and OptiDep tries to map incoming requests to the cheapest VMs that meets the required capacity which will result in maximizing the resource utilization. Our cost model does not prioritize large VMs as opposed to CostVNE. This will explain the case that, for 15 and 20 nodes, OptiDep outperforms CostVNE to obtain resource savings in terms of CPU that reaches 10% in the best cases. In consequence, we can say that OptiDep not only enables better cost savings but also maximizes the resource utilization.

OptiDep approach outperforms greatly the Vineyard approach. The resource savings can go up to 55 % for 30 nodes. When the number of nodes is small e.g. 5 nodes, we can see that the two approaches have very close results. However, as we increase the number of nodes, we see that the performance of Vineyard is degrading e.g. for 5 nodes the resource savings is hardly 5 % but for 20 nodes it increases to 36% to go up to 55 % for 30 nodes. Moreover, the gap between the two approaches is getting bigger with dense graphs. This result highlights the fact that OptiDep as an exact approach, enables better use of resources compared to Vineyard which performs its rounding decisions after mapping the abstract nodes in parallel for all the abstract nodes without taking into account the fact that the selection of one abstract node may affect others' which results in sub-optimal use of resources. Besides, we can conclude from the figure that the CPU utilization decreases for both approaches as the graph gets denser. This is due to the fact that, with dense graphs, the link demand increases and thus, in most cases, the nodes that are linked to edges with sufficient capacities are selected rather than the nodes that maximize the resource utilization.

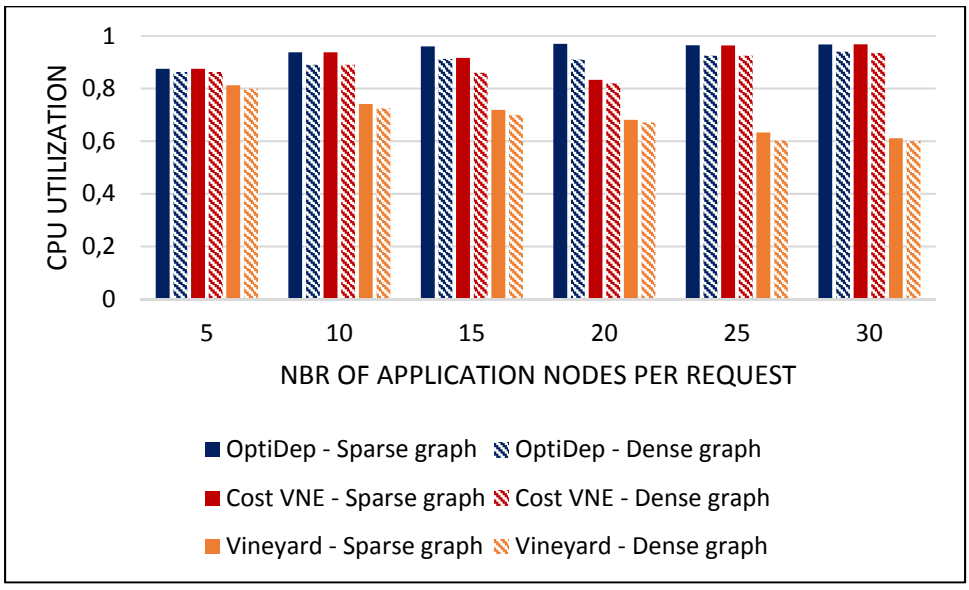

Figure 5.14 CPU utilization versus the number of application nodes per request

- *Memory*

We can see in Figure 5.15 the same trends as the CPU utilization. In general, OptiDep and CostVNE approaches have very similar results since the two approaches try to minimize the resource utilization as we have said previously. Similar to CPU utilization results, we can see from the figure that OptiDep enables better memory utilization. For 20 nodes, it saves 20 % memory resources compared to CostVNE.

OptiDep saves up to 78% compared to Vineyard when the topology has 30 nodes. We can see from the figure that when the number of nodes is small, OptiDep and Vineyard tend to have similar results but as we increase the number of nodes, Vineyard performance is degraded whereas our approach always finds the optimal solution as the number of nodes increases and as the graph gets denser.

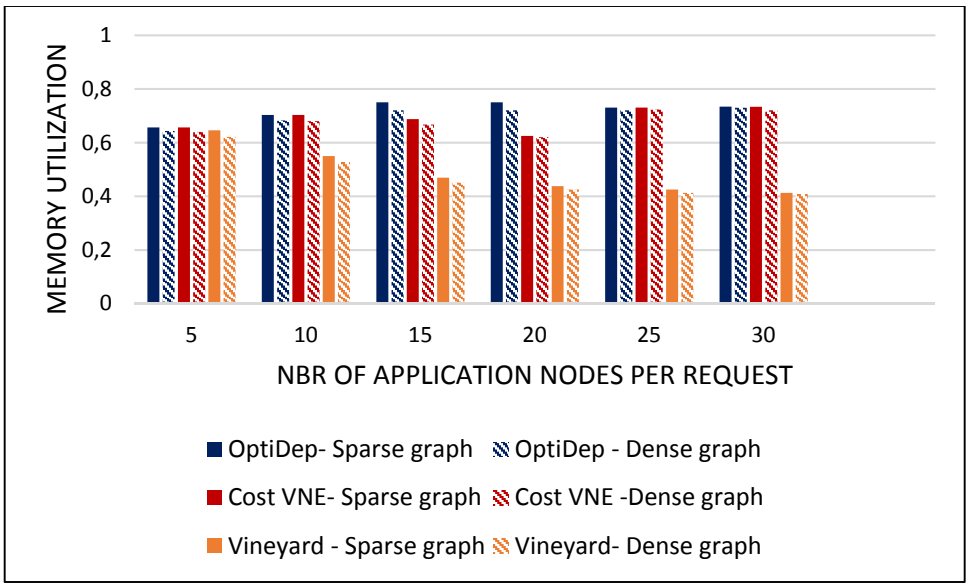

Figure 5.15 Memory utilization versus the number of application nodes per request

- *Bandwidth*

From Figure 5.16, we can conclude that the three approaches have very close results. The main reason behind this is the fact that OptiDep, CostVNE, and Vineyard use multicommodity flow problem to perform the link assignment. We can see from the figure that Vineyard performance is less than OptiDep and CostVNE due to its relaxation techniques.

Moreover, the figure depicts that, as the number of application nodes increases and as the graph gets denser, the link utilization increases.

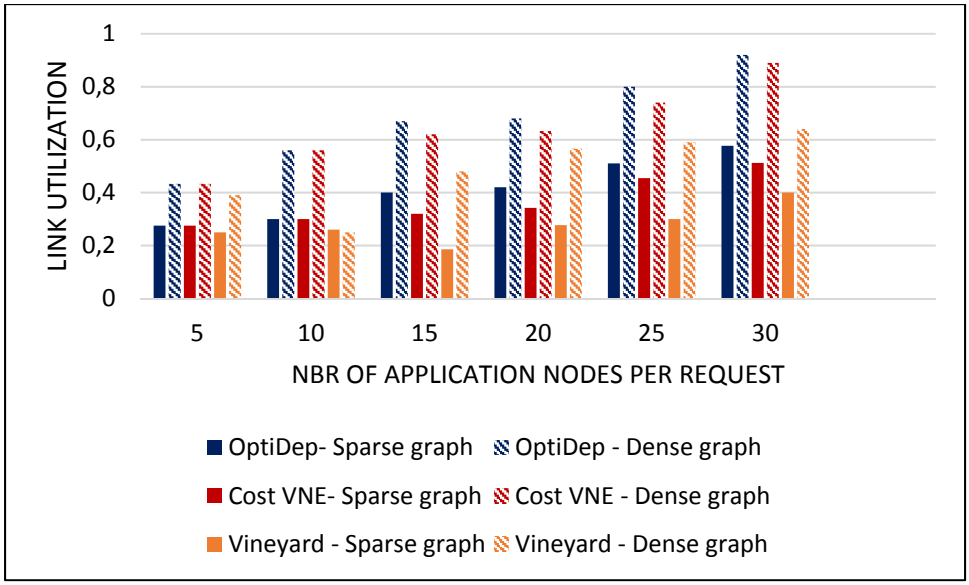

Figure 5.16 Bandwidth utilization versus the number of application nodes per request

5.3.5.3 Acceptance ratio

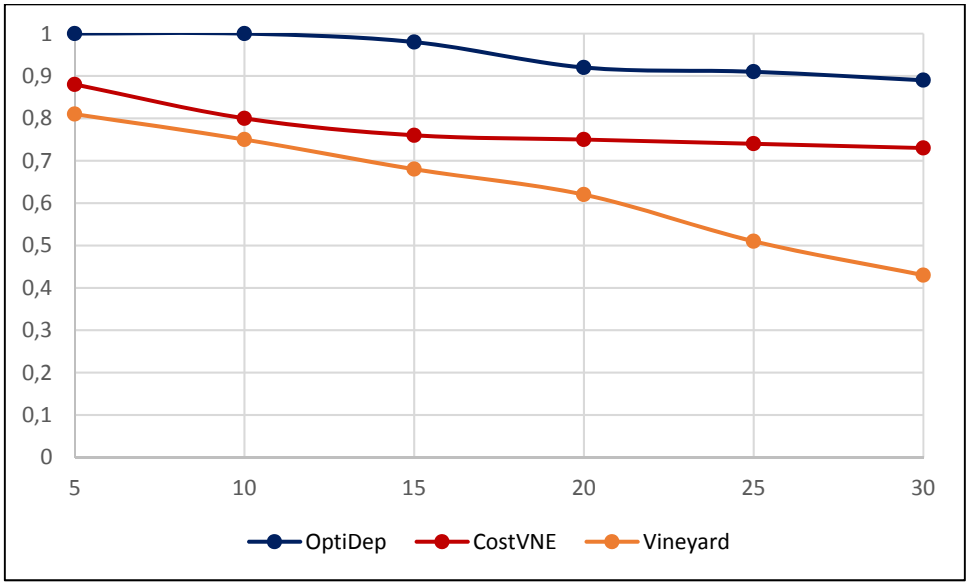

Figure 5.17 Acceptance ratio versus the number of application nodes per request

Figure 5.17 depicts the acceptance ratio when varying the number of application nodes per request. We can see that OptiDep has higher acceptance ratio than CostVNE and Vineyard. This is due to the fact that these two latter approaches (e.g. CostVNE and Vineyard) do not consider the networking demand between the local-based component and cloud-based components resulting in less accepted solutions.

5.3.5.4 Computation time

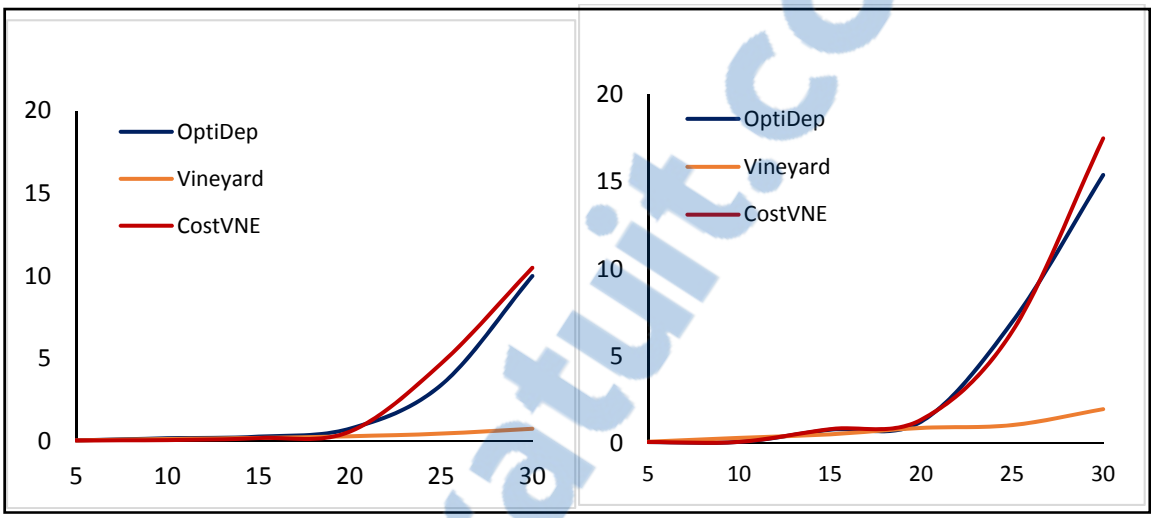

Figure 5.18 Computation time versus the number of application nodes per request

OptiDep and CostVNe are exact approaches that provide optimal mapping solutions. However, since MILP problems are hard to solve, their complexity grow exponentially with their size compared to heuristic solutions such as Vineyard that enable sub-optimal solutions but with reduced delays.

As shown in the Figure 5.18, the computation time needed for the mappings grows exponentially with the increasing number of application nodes per request as it reaches 10 seconds when the number of application nodes is 30 per request in case of a sparse graph and 15 seconds in case of a dense graph for the OptiDep approach. We can see nevertheless that OptiDep outperforms CostVNE in terms of computation time.

To sum up, we can conclude that heuristic mapping solutions are more suitable for large-scale networks.

5.3.6 Discussion

Evaluation results confirm that OptiDep enables better cost savings up to 29% on average after 26 scenarios which is a significant number compared to CostVNE. This is due to the fact that CostVNE tries to leave as much free capacity as possible for incoming requests regardless of the cost of the VMs and links, which will result in higher rental costs.

Results also confirm that OptiDep not only minimizes rental costs but also maximizes the resource utilization. In addition, OptiDep considers smart home specific constraints such as the communication delay between local-based and cloud-based components which is highlighted by the results of the acceptance ratio showing that CostVNE enables higher acceptance ratio since it does not take into account the bandwidth requirement between the local-based and cloud-based components. Finally, OptiDep is an exact approach that requires more time to map requests than heuristic algorithms. However, the matching process introduced before the request mapping enables to reduce the size of the virtual graph and, in our specific context, OptiDep is the most suitable solution.

Conclusion

In this chapter, we have presented at first the implementation of the proposed architecture. After that, we have shown an example of modeling dependencies of application requirements using regression algorithms. Results have shown that this method enables building the user dependency model by efficiently discovering dependencies and modeling the relationship between application requirements and QoS classes.

Finally, we have presented the simulation results of our approach OptiDep compared to other approaches considering the mapping costs, resource utilization, acceptance ratio and computation time. Results have shown that our approach outperforms CostVNE and Vineyard algorithms in terms of cost savings (29% compared to CostVNE and 76% compared to Vineyard) and resource utilization (up to 20% compared to CostVNE and 55% compared to Vineyard).

GENERAL CONCLUSION

With the actual growing popularity of the Internet of Things (IoT) and of robotics, smart home and home automation are considered as the next big opportunity. World leading technology companies like Ericsson, Google, Amazon and Apple are competing to provide better smart home applications. At the same time, home automation applications are becoming more diverse and resource demanding.

Cloud computing, as it offers on-demand, pay-per-use and scalable computing resources (e.g. CPU, memory, storage) can be viewed as a promising solution for hosting smart home applications.

The optimal integration of smart home vertical applications with cloud computing is challenging. In particular, allocating more resources than required when virtualizing applications in the cloud will incur inevitable unnecessary costs especially in this utility environment where allocated resources are charged by cloud providers to application owners. The virtualization process has to allocate proper resources while minimizing infrastructure costs. In addition, manually deploying such complex services can be expensive, time-consuming and error-prone.

This research has addressed two major challenges in the virtualization of smart home applications. The first challenge is how to map home applications to cloud resources in order to minimize costs and maintaining the required Quality of Service, and the second challenge is how to automatically deploy these applications onto the cloud.

Most of the prior work tried to map application components to virtual machines which may result in suboptimal solutions since they haven't considered the entire placement problem from the application layer down to the physical layer. Besides, they have not considered the pricing model defined by the current cloud providers. Furthermore, no existing solution has considered

the specific characteristics of home applications which are fundamentally different regarding other web or mobile applications.

The contributions of this reearch are:

- We proposed a Mixed Integer Linear Programming (MILP) optimization model to minimize mapping costs while maximizing resource utilization and maintaining the required Quality of Service (QoS) of applications to be deployed. This solution takes into account the whole placement problem from the application to the infrastructure layer;

- We considered the pricing model of leading Cloud providers, as well as the constraints and characteristics specific to home automation applications;

- We designed a system that automates the deployment of complex distributed applications onto Cloud;

- We proposed a method to model dependencies through statistical regression analysis between compute, network requirements and QoS classes to help the user define its specifications;

To compute an optimal mapping of the application graph into the infrastructure graph, we proposed OptiDep, an MILP based solution, to the application placement problem. We evaluated the performance of our approach compared to existing approaches. In our simulations, OptiDep has proven cost minimization for up to 29 % compared to another exact approach and more than 76 % compared to a heuristic-based solution and improves significantly resource utilization. We have implemented a system to automatically deploy complex services onto the cloud environment. Such a system has been integrated with OpenStack.

Future work

In our scenario, when the number of smart home applications and the number of smart home users are both small, OptiDep is the most suitable solution. However, as the smart home market is growing exponentially, we believe that, in future, OptiDep can be less efficient in large scale scenarios. Thus, it can be regarded as the first solution to the application placement problem in the smart home context and can be considered as an optimal bound to evaluate future approaches.

Besides, our application placement problem could be extended to include a placement order model where components with greater resource utilization such as a database component have higher priority and are placed at first to ensure the required availability.

In addition, the reliability issue in this thesis has not been addressed where a single service instance will not be sufficient but a set of service replicas. In future, we can extend our work to address the Facility Location Problem to find out the best strategy to place these replicas.

Moreover, hiding the heterogeneity of smart home devices coming from different smart home providers to offer a wider range of applications is an issue that has not been addressed in this thesis. This can be resolved by virtualizing smart home gateways for the different vendors and optimizing their placement on the cloud.

Furthermore, providing the required QoS is considered in this thesis by responding to computing and networking requirements of services to be deployed. However, in practice, other considerations may be taken into account such as real-time VM interaction which can result in QoS degradation and need for that employing VM migration and re-allocation techniques based on QoS measurements. Therefore, we intend to include dynamic scaling and migration functionalities to maintain the required quality of service (QoS).

Our optimal virtualization system ensures an automatic deployment of complex services in the cloud environment. Currently, the system does not handle failures. We intend to improve the system by introducing fault-tolerant and resilient mechanisms.

APPENDIX I

EXAMPLE OF A DEPLOYABLE STACK

```
{
  "Applications": [
    {
      "Name": "Video Monitoring",
      "Modules": [
        "MD",
        "IS",
        "UM",
        "UN"
      ],
      "Virtual": [
            {
        "m1.tiny": []
      },
      {
        "m1.small": [
          "MD",
         ]
      },
      {
        "m1.medium": [
            "IS"
        ]
      },
      {
        "m1.large": [
          "UM"
        ]
      },
```

EXAMPLE OF A DEPLOYABLE STACK (continued)

```
    {
      "m1.xlarge": []
    }
  ],
  "Infrastructure": [
    {
      "ets-blade-7": [
        "FD"
      ]
    },
    {
      "ets-blade-19": [
        "IS",
        "UM",
        "UN"
      ]
    },
    {
      "SEPIA": []
    }
  ],
  "Protocol": [
    {
      "MD": "HTTP",
      "IS": "HTTP",
      "UM": "HTTP",
      "UN": "HTTP"
    }
  ]
 }
]
```

APPENDIX II

EXAMPLE OF A MASTER DEPLOYMENT TEMPLATE

```
heat_template_version: 2015-04-30

description: Master template that installs composed application

parameters:
  image:
    type: string
    label: Image name or ID
    description: Image to be used for server. Please use an Ubuntu based image.
    default: ubuntu server 14.04
  flavor:
    type: string
    label: Flavor
    description: Type of instance (flavor) to be used on the compute instance.
    default: m1.small
  key:
    type: string
    label: Key name
    description: Name of key-pair to be installed on the compute instance.
    default: demo-key
  private_network:
    type: string
    label: Private network name or ID
    description: Network to attach server to.
    default: net
```

EXAMPLE OF A MASTER DEPLOYMENT TEMPLATE (continued)

```
public_network:
    type: string
    label: Public network name or ID
    description: Public network to attach server to
    default: external
resources:
  Service1:
    type: Service1.yaml
    properties:
      image: { get_param: image }
      flavor: { get_param: flavor }
      key: { get_param: key }
      private_network: { get_param: private_network }
  Service2:
    type: Service2.yaml
    properties:
      image: { get_param: image }
      flavor: { get_param: flavor }
      key: { get_param: key }
      private_network: { get_param: private_network }
  floating_ip:
    type: floating_ip.yaml
    properties:
      port: { get_attr: [Service2, port] }
      public_network: { get_param: public_network }
outputs:
  ip:
    description: The public IP address to access Service2.
    value: { get_attr: [floating_ip, ip] }
```

APPENDIX III

EXAMPLE OF A DEPLOYMENT TEMPLATE OF AN APPLICATION COMPONENT

```
heat_template_version: 2015-04-30
description: Simple template to deploy a single compute instance
parameters:
  image:
    type: string
    label: Image name or ID
    description: Image to be used for compute instance
    default: test1
  flavor:
    type: string
    label: Flavor
    description: Type of instance (flavor) to be used
    default: m1.small
  public_network:
    type: string
    label: Public network name or ID
    description: Public network with floating IP addresses
    default: external
  key:
    type: string
    label: key name
    description: key to be used
    default: demo-key
```

EXAMPLE OF A DEPLOYMENT TEMPLATE OF AN APPLICATION COMPONENT (continued)

```
resources:
  web_server_security_group:
    type: OS::Neutron::SecurityGroup
    properties:
      name: web_server_security_group
      rules:
        - protocol: tcp
          port_range_min: 80
          port_range_max: 80
        - protocol: tcp
          port_range_min: 443
          port_range_max: 443
        - protocol: icmp
        - protocol: tcp
          port_range_min: 22
          port_range_max: 22
  private_network:
    type: OS::Neutron::Net
  private_subnet:
    type: OS::Neutron::Subnet
    properties:
      network_id: { get_resource: private_network }
      cidr: 10.0.0.0/24
      dns_nameservers:
        - 8.8.8.8
  router:
    type: OS::Neutron::Router
    properties:
```

EXAMPLE OF A DEPLOYMENT TEMPLATE OF AN APPLICATION COMPONENT (continued)

```
    image: { get_param: image }
    flavor: { get_param: flavor }
    key_name: { get_param: key }
    networks:
     - port: { get_resource: my_port }
    user_data_format: RAW
    user_data: |
     #cloud-config
     runcmd:
      - sudo su
                    - ./script_service1.sh
  floating_ip:
   type: OS::Neutron::FloatingIP
   properties:
    floating_network: { get_param: public_network }

  floating_ip_assoc:
   type: OS::Neutron::FloatingIPAssociation
   properties:
    floatingip_id: { get_resource: floating_ip }
    port_id: { get_resource: my_port }
outputs:
  instance_name:
   description: Name of the instance
   value: { get_attr: [ my_instance, name ] }
  instance_ip:
   description: IP address of the deployed instance
   value: { get_attr: [ floating_ip, floating_ip_address ] }
```

LIST OF REFERENCES

Andersen, David G. 2002. « Theoretical approaches to node assignment ». *Computer Science Department*, p. 86.

AWS. 2017a. « AWS pricing ». < https://aws.amazon.com/fr/ec2/pricing/ >.

AWS. 2017b. « Iot platform, how it works ». < https://aws.amazon.com/iot-platform/how-it-works/ >.

Botero, Juan Felipe, Xavier Hesselbach, Michael Duelli, Daniel Schlosser, Andreas Fischer et Hermann De Meer. 2012. « Energy efficient virtual network embedding ». *IEEE Communications Letters,* vol. 16, nº 5, p. 756-759.

Butt, Nabeel Farooq, Mosharaf Chowdhury et Raouf Boutaba. 2010. « Topology-awareness and reoptimization mechanism for virtual network embedding ». In *International Conference on Research in Networking*. p. 27-39. Springer.

Chowdhury, Mosharaf, Muntasir Raihan Rahman et Raouf Boutaba. 2012. « Vineyard: Virtual network embedding algorithms with coordinated node and link mapping ». *IEEE/ACM Transactions on Networking (TON),* vol. 20, nº 1, p. 206-219.

Cloud, Amazon Elastic Compute. 2011. « Amazon web services ». *Retrieved November,* vol. 9, p. 2011.

Cloud, One. 2013. « Advantages One Cloud ». < http://www.onecloudsol.com/virtualization.html >.

Davis, Lawrence. 1991. « Handbook of genetic algorithms ».

Derhamy, Hasan, Jens Eliasson, Jerker Delsing et Peter Priller. 2015. « A survey of commercial frameworks for the Internet of Things ». In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*. p. 1-8. IEEE.

Dorigo, Marco, Mauro Birattari et Thomas Stutzle. 2006. « Ant colony optimization ». *IEEE computational intelligence magazine,* vol. 1, nº 4, p. 28-39.

Dubois, Daniel J, et Giuliano Casale. 2016. « OptiSpot: minimizing application deployment cost using spot cloud resources ». *Cluster Computing,* vol. 19, nº 2, p. 893-909.

Glover, Fred, et Manuel Laguna. 2013. *Tabu Search∗*. Springer.

Gubbi, Jayavardhana, Rajkumar Buyya, Slaven Marusic et Marimuthu Palaniswami. 2013. « Internet of Things (IoT): A vision, architectural elements, and future directions ». *Future generation computer systems,* vol. 29, nº 7, p. 1645-1660.

Houidi, Ines, Wajdi Louati, Walid Ben Ameur et Djamal Zeghlache. 2011. « Virtual network provisioning across multiple substrate networks ». *Computer Networks,* vol. 55, nº 4, p. 1011-1023.

Houidi, Ines, Wajdi Louati et Djamal Zeghlache. 2008. « A distributed virtual network mapping algorithm ». In *Communications, 2008. ICC'08. IEEE International Conference on*. p. 5634-5640. IEEE.

Igarashi, Yuichi, Kaustubh Joshi, Matti Hiltunen et Richard Schlichting. 2014. « Vision: Towards an extensible app ecosystem for home automation through cloud-offload ». In *Proceedings of the fifth international workshop on Mobile cloud computing & services*. p. 35-39. ACM.

Kennedy, James. 2011. « Particle swarm optimization ». In *Encyclopedia of machine learning*. p. 760-766. Springer.

Lee, Kiho, Ronnie D Caytiles et Sunguk Lee. 2013. « A Study of the Architectural Design of Smart Homes based on Hierarchical Wireless Multimedia Management Systems ». *International Journal of Control and Automation,* vol. 6, nº 6, p. 261-266.

Lindsay, Greg, Beau Woods et Joshua Corman. 2016. « Smart Homes and the Internet of Things. URLhttps ». *otalliance. org/system/files/files/initiative/documents/smart_homes_0317_web. pdf*.

Lischka, Jens, et Holger Karl. 2009. « A virtual network mapping algorithm based on subgraph isomorphism detection ». In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. p. 81-88. ACM.

Manvi, Sunilkumar S, et Gopal Krishna Shyam. 2014. « Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey ». *Journal of Network and Computer Applications,* vol. 41, p. 424-440.

Meindl, Bernhard, et Matthias Templ. 2012. « Analysis of commercial and free and open source solvers for linear optimization problems ». *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*.

Mell, Peter, et Tim Grance. 2011. « The NIST definition of cloud computing ».

Microsoft. 2017a. « IoT hub service ». < https://azure.microsoft.com/en-us/services/iot-hub/>.

Microsoft. 2017b. « Microsoft azure pricing ». < https://azure.microsoft.com/en-us/pricing/ >.

Moore, Reagan W, et Chaitan Baru. 2003. *Virtualization services for data grids*. John Wiley & Sons.

Mosteller, Frederick, et John Wilder Tukey. 1977. « Data analysis and regression: a second course in statistics ». *Addison-Wesley Series in Behavioral Science: Quantitative Methods*.

Padmavathi, G. 2016. « Internet of Things-An Overview ». *World Scientific News,* vol. 41, p. 227.

Pandey, Suraj, Linlin Wu, Siddeswara Mayura Guru et Rajkumar Buyya. 2010. « A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments ». In *Advanced information networking and applications (AINA), 2010 24th IEEE international conference on*. p. 400-407. IEEE.

Patierno, Paolo. 2015. « AN IOT PLATFORMS MATCH : MICROSOFT AZURE IOT VS AMAZON AWS IOT ».

RightScale. < http://www.rightscale.com/ >.

Rouse, Margaret. 2016. « Exploring data virtualization tools and technologies ».

Samsung. 2017. « SmartThings ». < https://www.smartthings.com/ >.

Sefraoui, Omar, Mohammed Aissaoui et Mohsine Eleuldj. 2012. « OpenStack: toward an open-source solution for cloud computing ». *International Journal of Computer Applications,* vol. 55, nº 3.

Wang, Shiqiang, Murtaza Zafer et Kin K Leung. 2017. « Online Placement of Multi-Component Applications in Edge Computing Environments ». *IEEE Access,* vol. 5, p. 2514-2533.

Whitmore, Andrew, Anurag Agarwal et Li Da Xu. 2015. « The Internet of Things—A survey of topics and trends ». *Information Systems Frontiers,* vol. 17, nº 2, p. 261-274.

Wolf, Brain. 2009. « Cloud Computing five layer model ». < http://www.bluelock.com/blog/cloud-computing-a-five-layer-model/ >.

Yu, Minlan, Yung Yi, Jennifer Rexford et Mung Chiang. 2008. « Rethinking virtual network embedding: substrate support for path splitting and migration ». *ACM SIGCOMM Computer Communication Review,* vol. 38, nº 2, p. 17-29.

Zhang, Qi, Lu Cheng et Raouf Boutaba. 2010. « Cloud computing: state-of-the-art and research challenges ». *Journal of internet services and applications,* vol. 1, nº 1, p. 7-18.