Content Delivery Networks as a Service (CDNaaS)

Louiza Yala

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1
COMUE **U**NIVERSITE **B**RETAGNE **L**OIRE

Ecole Doctorale N°601
Mathèmatique et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

« Louiza Yala »

« Content Delivery Network as a Service (CDNaaS) »

Thèse présentée et soutenue à RENNES , le 23 Novembre 2018
Unité de recherche : IRISA

Rapporteurs avant soutenance :
Hind Castel, Professeur Télécom Sud Paris
Abdelhamid Mellouk, Professeur Université de Paris 12

Composition du jury :

Hind Castel, Professeur, Télécom Sud Paris. *Rapporteur*
Adlen Ksentini, MDC-HDR, Eurecom. *Directeur de thèse*
Bertrand Mathieu, Ingénieur de recherche, Orange. *Examinateur*
Abdelhamid Mellouk, Professeur Université de Paris 12. *Rapporteur*
Daniel Négru, MDC-HDR, Université de Bordeaux. *Examinateur*
César Viho, Professeur, Université de Rennes 1. *Examinateur*

Dir. de thèse : Adlen Ksentini, MDC-HDR, Eurecom.

TABLE OF CONTENTS

Resumé en Français **1**

1 Thesis Introduction **7**
1.1 Motivation . . . 7
1.2 Contributions of the Thesis . . . 8
1.3 Organization of the Manuscript . . . 10

2 Context **13**
2.1 Introduction . . . 13
2.2 Content Delivery Network (CDN) . . . 14
2.3 Network Functions Virtualization (NFV) . . . 18
2.4 MEC in NFV environment . . . 20
2.5 Conclusion . . . 22

3 An architecture for on-demand service deployment and provision over a telco CDN **25**
3.1 Introduction . . . 25
3.2 State of the art . . . 26
3.2.1 ISP-CDN collaboration and Telco CDN . . . 26
3.2.2 Data center virtualization . . . 29
3.2.3 Network Functions Virtualization (NFV) . . . 29
3.3 A Framework for CDNaaS provision . . . 31
3.3.1 Features . . . 31
3.3.2 Architecture . . . 31
3.3.3 Implementation . . . 35
3.3.4 vCDN flavors . . . 36
3.3.5 Our framework in the context of the ETSI NFV-MANO specification . . . 36
3.3.6 Service deployment time . . . 37
3.4 Enabling technologies and their performance . . . 40
3.4.1 Virtualization technologies and testbed configuration . . . 41
3.4.2 Startup time comparison . . . 43
3.4.3 Performance of a generic HTTP service . . . 43
3.4.4 Performance of an HTTP video streaming service . . . 47

3.5 Conclusion . . . 53

4 Ressource management and VNF placement 55
4.1 Introduction . . . 55
4.2 State of the art . . . 57
4.3 Cost- and availability-aware VNFs placement context . . . 64
4.4 A QoE-aware virtual CPU resource allocation algorithm . . . 65
4.4.1 System model . . . 66
4.4.2 An algorithm for the minimum cost solution . . . 67
4.4.3 Performance evaluation . . . 68
4.5 A model for joint vCPU-to-VM allocation and VM placement . . . 70
4.5.1 Preliminaries . . . 71
4.5.2 Cost model . . . 71
4.5.3 Availability model . . . 72
4.5.4 Problem formulation . . . 73
4.6 Solving a relaxed version of the problem . . . 74
4.6.1 Deciding on the number of VMs to launch . . . 76
4.6.2 VM placement and vCPU distribution . . . 78
4.7 A Genetic Algorithm as an alternative approach . . . 80
4.8 Performance evaluation . . . 82
4.8.1 Cost vs. availability tradeoff . . . 83
4.8.2 Performance comparison . . . 85
4.9 A note on the relevance with ETSI NFV . . . 91
4.10 A note on network resource allocation . . . 92
4.11 Conclusion . . . 92

5 Ultra-reliable and low latency VNFs placement in a MEC-NFV environment 95
5.1 Introduction . . . 95
5.2 State of the art . . . 96
5.3 Problem formulation . . . 98
5.3.1 Latency model . . . 98
5.3.2 Availability model . . . 99
5.3.3 Cost model . . . 100
5.3.4 Objective . . . 101
5.4 A Genetic Algorithm for VM placement to guarantee an uRLLC system . . . 102
5.5 Numerical results . . . 103
5.6 Conclusion . . . 108

6 Conclusions and perspectives **109**
6.1 Results obtained during the thesis 109
6.2 Perspectives 110

Acronymes **113**

Publications from the thesis **115**

Bibliography **125**

LIST OF FIGURES

2.1 Content Delivery Network overview. 14
2.2 Layered architecture of a CDN [86] 15
2.3 Architectural components of a CDN [25] 16
2.4 System components of a delivery network of the CDN provider Akamai [82] . 18
2.5 Functional blocks of NFV architecture [80] 19
2.6 MEC system 21
2.7 MEC in NFV reference architecture [72] 21

3.1 CDNaaS architectural components [31]. 32
3.2 Testbed configuration. An OpenStack setup with two compute nodes was created. The OpenStack controller was collocated with one of them. A separate machine was hosting our CDNaaS orchestration services (customer API, SO, SIDR, RO), over which we carried out vCDN service deployment requests. The average RTT in each path was configured to d. 38
3.3 Time until a vCDN deployment on a single data center becomes fully operational, under different experimental settings. 39
3.4 Time until a vCDN deployment in a single data center becomes fully operational as a function of the number of VNF instances to be launched, when VM images are already cached. 40
3.5 Time until a vCDN deployment in a single data center becomes fully operational as a function of the number of VNF instances to be launched, when VM images are not cached. For the case of a remote data center, image transfer times dominate the overall service instantiation time. 40
3.6 HTTP request throughput for increasing numbers of parallel connections. We compare the use of docker and kvm and the impact of utilizing more CPU resources. 44
3.7 HTTP request throughput for various object sizes and for increasing numbers of concurrent connections. 45
3.8 Comparison of the response time distributions of a virtualized vs. a containerized HTTP server, for different numbers of concurrent connections. Parallel HTTP connections request a minimal embedded object (43-byte empty GIF file). 46

3.9 Response time distributions for different sizes of the requested objects. . . . 47
3.10 Response time distributions for a server hosted in a kvm VM as the number of available vCPUs scales, for different numbers of parallel HTTP connections. Clients request a minimal embedded object (43-byte empty GIF file). 47
3.11 Average QoE as a function of the number of parallel users accessing a HD video from a web server. Each point is the mean of a few hundreds of QoE samples (MOS values), presented with 95% confidence intervals. 50
3.12 Interruption frequency (in interruptions/minute) as server load grows. 50
3.13 Empirical CDF of the QoE of video samples for two different server loads when the service is hosted in a kvm virtual machine. 51
3.14 Probability that quality in a sample is acceptable (i.e., MOS > 3.6). 52

4.1 Comparison of a QoE-aware with a fixed resource allocation mechanism. Depending on its configuration, the baseline scheme may lead to under- or over-provisioning. Our QoE-driven approach, on the other hand, derives an assignment which minimizes cost respecting the quality constraint. 69
4.2 Execution time of our QoE-aware algorithm for increasing topology sizes (i.e., numbers of regions/POPs.) Each point is the mean value of 50 iterations. . . 70
4.3 Solution space. The straight lines represent cost and availability constraints. The filled boxes are the assignments selected as optimal by our TRAP algorithm under different policies, while the empty boxes are feasible but suboptimal solutions. 84
4.4 Availability and cost as a function of the selected policy. 84
4.5 Performance of different placement schemes for the same policy and increasing numbers of available PMs. 86
4.6 Performance of different placement schemes for the same configuration (number of available PMs) as a function of the applied policy (criteria weights). . . 87
4.7 Comparison between our two-step heuristic (TRAP) and our GA in terms of the cost objective for different policies (lower is better). 88
4.8 Comparison between our two-step heuristic (TRAP) and our GA in terms of the availability objective for different policies (higher is better). 88
4.9 Execution time of the GA as a function of the number of generations (G). . . 89
4.10 Execution time of our GA as a function of the pool size (S). 89
4.11 Performance of the GA as the size of the pool (S) increases. 89
4.12 Performance of the GA as the number of generations (G) increases. 89
4.13 Execution time as a function of the number of available PMs for different algorithms. 90

5.1 Availability and latency as functions of the selected policy. 104
5.2 MEC servers utilization as a function of the policy. 104
5.3 Evolution of cost with the latency-oriented policy. The straight line represents the cost constraint (budget). 104
5.4 Box-plot showing VMs' impact on the average latency of a service deployment. 104
5.5 Availability as a function of the number of VMs. The straight line represents the availability constraint. 104
5.6 Cost objective value as a function of the number of VMs. The straight line represents the cost constraint. 104
5.7 Comparison between our GGA and CPLEX in terms of the objective for a specific policy. 107
5.8 Execution time as a function of the number of PMs for our GGA and CPLEX. 107

LIST OF TABLES

3.1 Startup times for a kvm Debian image and a docker container 43
3.2 Chunk sizes (bytes) for our test H.264/AVC DASH video sequences. 45
3.3 Summary of results. QoE as a function of the number of parallel users (streams) accessing a video service hosted on a single-core kvm-based web server/cache VNF. 52

4.1 State of the art classification according to the criteria considered. 64
4.2 Summary of notation for QoE-aware vCPU resource allocation 67
4.3 Summary of notation for joint vCPU-to-VM allocation and VM placement . . . 75

RESUMÉ EN FRANÇAIS

Dans les réseaux de diffusion de contenu appelés *Content Delivery Network* (CDN), les serveurs et leur centre de données (data center) sont géographiquement distribués. De part leurs différents emplacements, ils distribuent selon les besoins, régionalement le service aux utilisateurs finaux. Ceci dans le but de fournir à ces derniers une haute disponibilité et de plus grandes performances. De nos jours, les CDNs servent une grande partie du contenu Internet, y compris (i) les objets web tels que le texte, les graphiques et les scripts, (ii) les objets téléchargeable tels que les fichiers multimédias (vidéos), les applications comme celles du commerce électronique, et (iii) le streaming en directe ou à la demande tels que Netflix, Amazon, etc. Les fournisseurs cherchent à exploiter la virtualisation en mettant à profit les plates-formes de virtualisation appelées *Network Functions Virtualization* (NFV) afin de déployer des ressources (caches, CPU...) sous forme de fonctions de réseau virtuelle (VNF) remplaçant les appareils physiques. La virtualisation des CDN (vCDN) peut amener plusieurs avantages, notamment le fait de pouvoir s'adapter aux nouvelles conditions de fonctionnement telles que la création, la suppression ou même la modification des composants CDN en les programmant seulement à distance (par exemple la planification d'une mise en service).

L'objectif de cette thèse est d'étudier et d'évaluer les avantages de la virtualisation des réseaux de diffusion de contenu (CDN) vidéo dans le cloud computing, afin de proposer des algorithmes de placement et d'instantiation de fonction virtuelle au sein d'un cloud fédéré qui est un cloud décentralisé dont le fonctionnement est uniforme et les services sont de types différents (indépendants).

Contributions et organisation de la thèse

Nous commençons par introduire dans le chapitre 2 le contexte de cette thèse en listant différents articles de la littérature qui abordent des sujets pertinents. Ces sujets traitent les réseaux de diffusion de contenu et l'évolution des fonctions de réseau virtualisées. Notons aussi que dans chaque chapitre de contribution, nous détaillons un état de l'art en rapport avec le problème localement traité.

1. Dans le chapitre 3 nous présentons notre première contribution : une architecture CDN

que nous avons implémenté et qui permet à un opérateur de réseau de virtualiser son infrastructure CDN (vCDN) et de la louer à des fournisseurs de contenu, cet act est appelé louer son logiciel (ou sa plateforme) en tant que service ou en anglais *Software as a Service* (SaaS). Dans notre cas la plateforme CDN en tant que service, appelée *CDN as a Service* (CDNaaS). Cette flexibilité permet à l'opérateur de proposer des solutions CDN moins couteuses et plus flexibles. Un exemple de flexibilité est que le virtuel CDN instancié par le fournisseurs de contenu (que nous désignons comme clients) puisse être couplé à d'autres infrastructures CDN. Ceci permet au client, en prenant avantage de la présence régionale de l'opérateur, d'utiliser l'infrastructure louée pour répondre à une augmentation du trafic dans certaines régions. D'un point de vu fournisseur CDNaaS, notre conception permet une utilisation plus efficace des ressources, comparé à un modèle de réservation de ressources moins dynamique (allocation statique des ressources). L'architecture logicielle proposée comprend différents blocs fonctionnels, communiquant via des interfaces bien spécifiées. Il est possible de dissocier les blocs de leur emplacement physique, ce qui permet au fournisseur CDNaaS d'exécuter chacun d'eux en tant que fonction virtuelles de façon autonome, sur sa propre infrastructure cloud. L'un des principaux blocs de notre architecture : le Service Orchestrator (SO), a pour but la coordination du déploiement des services vCDN. Après réception d'une requête d'un client, le SO déploie et place le VNF adéquat, prenant en compte les exigences du service, les spécifications de la demande et la qualité demandée par le client (incluse dans la demande du service), ainsi que la capacité opérationnelle du fournisseur. Les détails techniques sont pris en charge par le Resource Orchestrator (RO), tels que la gestion des adresses IP pour les instances déployées, mais aussi l'implémentation du service nécessaire (par exemple, configurer la planification du démarrage du serveur DNS afin de gérer de manière appropriée la géolocalisation ou la configuration des instances virtuelles du cache à partir des requêtes proxy de l'utilisateur vers les serveurs d'origine). Notons que cette politique peut différer selon le type de vCDN disponible chez l'opérateur.

Notre solution offre aux clients, une interface nord (Northbound RESTful API) (détailler dans le manuscrit chapitre 3), à travers laquelle ils demandent le déploiement d'un vCDN sur le telco Cloud. En utilisant l'API, le client spécifie ses besoins en termes de nombre de demande qu'il souhaite couvrir par région, c'est à dire qu'il donne le nombre d'utilisateurs final à qui il souhaite offrir le service dans une région donnée. Il exprime aussi ses contraintes de performance, en terme de qualité moyenne souhaitée et un temps de réponse désiré, etc. L'interface nord lui assure un contrôle suffisant au niveau du service, en faisant abstraction du réseau interne et des détails d'infra-

structure. Quand le service est déployé avec succès, l'API renvoie un point d'entrée à l'instance vCDN afin que le client puisse accéder/gérer son déploiement.

2. Dans le but d'avoir une allocation optimale des ressources, notre proposition (deuxième contribution de cette thèse qui correspond au chapitre 4) est en mesure de combiner les informations fournies lors de la demande par le client (par exemple une éstimation du nombre d'utilisateur/flux vidéo par région) avec les données du réseau et de l'infrastructure de calcul. Il est important de connaitre les capacités des dites technologies et de se baser sur la relation entre la charge du service et l'expérience de l'utilisateur. Ceci afin de décider d'une bonne allocation des ressources, un bon placement et une gestion en temps réel. Tout cela en minimisant le coût opérationnel du point de vu fournisseur et en offrant le niveau de service convenu avec le client. Une partie des résultats que nous présentons dans ce manuscrit sont l'aboutissement de ces derniers points, à savoir économiser les ressources tout en satisfaisant la demande du client. Nous proposons de quantifier la relation entre la charge du serveur et l'expérience de l'utilisateur. Pour ce faire, nous avons mesuré les performances d'un serveur vidéo HTTP utilisant une seule CPU virtuelle en présence de plusieurs sessions vidéo parallèles. Une vCPU dans notre système est une l'unité de la ressource qui correspond à un coeur CPU. L'approche est centrée utilisateur car elle se base sur la qualité de service que le client souhaite offrir à ses utilisateurs finaux. D'un point de vue spécifique nous avons quantifié empiriquement la relation entre la charge "*workload*" du service et la qualité d'expérience *QoE* du service vidéo. À travers les résultats obtenus, nous remarquons un temps de lecture de la vidéo avec une QoE supérieure à un seuil bien défini. Ceci peut être traduit dans un Service Level Agreement (SLA) [1] d'un format diffèrent, où le client peut, s'il le souhaite, assurer que les utilisateurs finaux bénéficient d'un niveau de QoE acceptable pour plus d'un pourcentage spécifique du temps de visionnage de la vidéo. Les différents tests effectués ont permis de mieux comprendre le potentiel et les capacités des technologies de virtualisation vis-à-vis de notre vision du *CDNaaS* et à les utiliser pour concevoir des stratégies appropriées, telles que la détermination de façon optimale du nombre de ressources à déployer et leur configuration. Grâce à ces stratégies de virtualisation, il est possible de répondre à la demande des clients et d'élaborer des politiques de prix pour le service CDN virtuel offert.

Dans cette partie de la contribution nous avons modélisé le problème d'allocation de ressources en problème d'optimisation, nous proposons ensuite un algorithme pour

1. Engagement de niveau de service, c'est un contrat dans lequel un fournisseur de contenu (ou de service) s'engage à fournir un ensemble de services à un ou plusieurs clients.

le résoudre. Nos résultats de simulation montrent l'importance de la *QoE-awareness* pour efficacement aborder le compromis entre la qualité de service et le coût.

3. Par la suite, dans la troisième contribution (suite du chapitre 4), nous proposons une formule d'optimisation du problème de placement des instances virtuelles (dans notre cas vCPUs et VMs) dans des machines physiques (PMs). Nous proposons un modèle pour l'allocation conjointe de vCPU-à-VM et le placement des VMs dans les PMs. Nous nous sommes plus précisement intéressé aux objectifs de la disponibilié du service et du coût qu'engendre un déploiement d'un service CDN qui sont des objectifs contradictoire. Ces critères nous ont conduit à proposer une formulation multi-objectif pour les deux problèmes. Le problème de placement de virtuelles machines est connu pour sa difficulté de résolution. C'est pour cela que nous avons traité ce problème multi-objectif en résolvant sa version relaxée à l'aide d'un algorithme que nous proposons. Nous avons d'abord trouvé le nombre dit *optimal* de VMs ensuite en prenant ce dernier comme entrée, l'algorithme trouve le nombre adéquat de PM et distribue uniformément les vCPUs parmi eux. En assouplissant les hypothéses du problème, nous avons proposé des algorithmes heuristiques polynomiaux pour le résoudre. Nos expériences montrent que les solutions obtenus par notre algorithme sont très proche de l'optimal, tout en étant plus rapide qu'un algorithme exact fourni par le solveur commercial d'IBM CPLEX [2]. De plus, nous avons comparé notre solution avec deux approches de placement de base. Les résultats obtenus ont largement démontré la supériorité de notre solution vis à vis des deux approches de base.

   Nous avons aussi comparé notre méthode avec une méta-heuristique : *l'algorithme génétique*, inspiré de l'état de l'art [106], que nous avons défini comme solution alternative, les résultats de ce dernier se rapprochent de l'optimal CPLEX et des résultats obtenus par notre heuristique.

4. Dans l'avant dernier chapitre 5 qui correspond à la dernière contribution nous avons étudié la manière avec laquelle nous pouvions utiliser nos précédents modèles dans un réseau mixte à savoir faire du placement VNF dans un cloud fédéré et dans un edge cloud, prenant les différents avantages de l'un et de l'autre. L'un des avantages des edge cloud est le temps de latence ultra-faible appelé *ultra Reliable Low-Latency Communications* (uRLLC). De part cet avantage, nous adaptons dans le chapitre 5, notre modèle afin de prendre en compte le edge cloud. Notre méthode permet d'abord de classer les ressources cloud en se basant sur leur latence d'accès au service ou à

2. http ://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

l'application. Ensuite en utilisant cette classification nous avons proposé une formulation du problème de placement, où les objectifs sont la minimisation de la latence et la maximisation de la disponibilité du service. Nous avons adapté l'algorithme génétique précédemment cité et comparé avec les résultats du solveur CPLEX. Nos résultats ont montré que notre solution est proche de l'optimal tout en minimisant le temps pour trouver la solution.

5. Nous concluons ce manuscrit par le chapitre 6, dans lequel nous résumons nos travaux en rappelant l'objectif de cette thèse, la problématique traitée et comment nous y avons répondu. Nous finissons par illustrer quelques perspectives de travaux futurs.

CHAPTER 1

THESIS INTRODUCTION

1.1 Motivation

With the emergence of popular video-streaming platforms that offer online content services for video such as Amazon and Netflix, Content Delivery Network (CDN) will carry more than 70% of the global Internet traffic by 2021 [15]. This new paradigm will inspire the internet architectures to go beyond this problem. This is especially due to the ever-increasing user expectations at low cost making the delivery of high Quality of Experience (QoE) and high user satisfaction for multimedia services more important than ever.

In particular, the problem is that in one hand content distributed over the top creates a bidirectional dependence between CDN and network providers: CDN customers, and, in turn, end users, depend on the underlying network infrastructure, and user experience is affected by its conditions. Network operators, on the other hand, are vulnerable to the traffic dynamics caused by time-varying shifts in content demand. At the same time, network providers wish to take advantage of their regional presence and proximity to the end users to enter the content delivery market.

This has led to the emergence of telco CDNs, which are content delivery infrastructures owned and operated by network providers. In this case, the network operator installs data centers at its points of presence (or other points strategically located in its network) and offers a CDN service advertising high-performance delivery due to user proximity. From the perspective of a content provider, traditional and telco CDNs are not competitive, but, rather, complementary services: a telco has the advantage of proximity to end users, but typically this is limited to specific geographic locations where it has presence. On the contrary, traditional CDN players compete offering a global service.

To achieve the emergence of the Telco Cloud, existing technologies such as Network Functions Virtualisation (NFV) and Software Defined Networking (SDN) arised to satisfy both Internet Service Provider (ISP) and Content Provider (CP). In this thesis we study the

role of a virtual Content Delivery Network (vCDN) in improving the end-users QoE while saving on service providers' costs and offering high service availability. In the next section, we outline our contributions in building a system that creates and manages virtual instances of CDN, whereby CDN Virtual Network Functions (VNF) are instantiated and strategically placed over a telco cloud, close to the end-users.

1.2 Contributions of the Thesis

The main objective of this thesis is to study and evaluate the benefits of virtualizing the CDN by means of cloud computing technologies. For this purpose several issues have been addressed. The significant contributions of the thesis are:

1. First, we propose an architecture which allows a network operator to virtualize its CDN infrastructure and lease it to content providers on demand [30, 32]. Our approach, which is in line with current standardization efforts on Network Functions Virtualization, improves on resource management and service provision flexibility compared to traditional telco CDNs. It can be viewed as an evolution of the telco CDN model, thus having the potential of empowering the role of telecom operators in the content delivery value chain. In particular our design features a Representational State Transfer (REST) Application Programming Interfaces (API) which enable content providers to lease virtual CDN resources on demand. Our scheme involves multiple blocks, communicating via well-specified interfaces. This decouples their operation from any physical location, allowing the Content Delivery Network as a Service (CDNaaS) provider to execute any of these blocks autonomously as virtual functions over its own cloud infrastructure. One of the most significant advantages of our design is to expose open APIs to customers, but also among the components of our architecture, and to design with extensibility in mind, so that our scheme can be extended towards a generic Any-as-a-Service model. In this first contribution we also carried out experiments to explore the capabilities and the limitations of the virtualization technologies that we apply by comparing *kvm* Virtual Machine (VM) with *Docker* containers.

2. Second, since our scheme allows content providers to express service demand specifications and QoE constraints, we used the results from the previous contribution and these specifications to derive resource placement algorithms that aim to optimally satisfy user demand and save on operators' cost. In particular the placement algorithm can be used to decide on an initial virtual resource allocation for the specific vCDN instance. We were motivated by the fact that QoE-awareness is critical in the process of deciding on an initial virtual resource allocation for the specific vCDN instance. An

allocation of virtual resources for a vCDN deployment request may lead to situations where user demand cannot be satisfied with the desirable quality (e.g., when the number of CPU resources allocated is not capable of handling the load of many parallel video streams), or, conversely, to over-provisioning, when more resources than necessary are utilized, leading to increased cost for the operator. In this direction we made the second contribution [107], wherein we carried out extensive measurements of an HTTP video service running on top of a virtualization platform to measure the capabilities of the latter from a QoE-centric viewpoint. Based on the obtained measurements we derived an expression of user experience as a function of video server load. This relationship is then used by a novel measurement-driven resource allocation algorithm which aims to cover a target end-user demand (i.e., parallel video streams per region) by minimizing the used computing resources, while respecting (customer) QoE and (operator) capacity constraints. Using simulations on a real Points of Presence (PoP) topology, we demonstrate the advantages of QoE awareness and the importance of our empirical measurements. Our algorithms achieve their goals while being efficient in terms of running time.

3. In the third contribution, we provide solutions to the distribution of the previous virtual resources among VMs. Then we solve the problem of distributing the latter on Physical Machines (PMs). For that we capture the conflicting objectives of service availability and deployment cost by proposing a multi-objective optimization formulation for the problem of joint compute resource allocation and VM placement [108]. Since the latter is computationally hard to solve efficiently, we proposed a polynomial-time heuristic algorithm to solve a relaxed version of the problem. We showed experimentally that the derived solutions are close to the optimal, while at the same time being significantly faster than an exact algorithm provided by a commercial solver. We also showed that our approach outperforms two baseline schemes (random placement and first-fit), as well as a Genetic Algorithm inspired from the state of the art which we consider as an alternative solution.

4. In the fourth and final contribution, we continue to focus on placement algorithms. The difference with the previous contribution is that we not only take into account centralized cloud servers, but we integrated edge servers. The previous placement algorithm aims to reduce deployment costs, increase the users' QoE, ensure service availability, etc., ignoring Mobile (or Multi-Access) Edge Computing (MEC) applications' constraints in terms of low latency. This could lead MEC application instances to be instead deployed to central cloud hosts, as the NFV Orchestrator NFV Orchestrator (NFVO) and its related placement algorithms do not differentiate between MEC

applications and other VNFs. We fill this gap by proposing a placement algorithm tailored to ultra-Reliable Low-Latency Communication (uRLLC) services in the context of a MEC in NFV environment. Our proposition classifies all cloud resources (central and edge cloud) according to their access latency to the user data plane. Then, the placement algorithm is formulated as an optimization problem, where the objectives are to minimize latency and to maximize service availability (reliability); these correspond to the main deployment criteria for uRLLC services. We provide solutions to this problem by proposing a suitable Genetic Algorithm, but also by using the CPLEX solver as a benchmark. Our experimental results show that our solutions are close to the optimal, while simultaneously it takes considerably less time to derive them than an exact algorithm provided by CPLEX. We have also shown how our method can reduce delays and still provide a highly-available service.

1.3 Organization of the Manuscript

The rest of the thesis is organized as follow.

Chapter 2 overviews the state of the art in CDNs identifying relevant CDN architectures. We investigate the integration of the virtualization in the cloud and CDN making them more flexible. We also highlight the ISP-CDN interactions and the emergence of telco CDN. Then we describe the emergence of NFV as a key technology in service delivery. Finally we outline the Mobile Edge Computing (MEC) in NFV environment by reporting a reference architecture.

In chapter 3, after an in-depth study of the state of the art in telco CDNs, data center virtualization and NFV we present our architecture for CDNaaS provision and an experimental evaluation of the capacity of candidate virtualization technologies which can support our vision.

Chapter 4 starts by an extensive analysis of the state of the art on VNF placement algorithms and a classification of these algorithms according to the criteria considered (placement contraints). Then, the chapter addresses issues of VNF placement and resource allocation by proposing a QoE-aware method to optimize the amount of vCDN resources allocated across the regions where the telco cloud operator has presence. The aim is to serve a content provider's vCDN deployment request under quality and capacity constraints. Our model for joint vCPU-to-VM allocation and VM placement among PMs is also presented in this chapter. The chapter also provides our proposition of the heuristic solution for a relaxed

version of the problem. We compare our heuristic with an adaptation of a Genetic Algorithm from the state of the art. Finally we present experimental results on the performance of our scheme compared with heuristics widely used in the literature, the aforementioned Genetic Algorithm, and an exact solution.

In chapter 5 we introduce uRLLC and the MEC environment. We present the state of the art for the particular problem of placing VNFs in a hybrid environment which merges centralized and edge resources in a deployment of a service. We present our formulation of the VNF placement problem for uRLLC and we show our solutions to this problem by adapting the precedent proposed Genetic Algorithm comparing its results with CPLEX solver. We also show how our method can reduce delays and still provide a highly-available service.

Finally, we conclude the thesis in chapter 6 where we also present perspectives and future directions for our work.

CHAPTER 2

CONTEXT

2.1 Introduction

The cloud computing is gaining high momentum for hosting and delivering services over the Internet. There are three main cloud computing service models: Infrastructure as a service(IaaS), platform as a Service (PaaS) and software as a service (SaaS). The key issue in IaaS delivery is how to efficiently virtualize the computing and storage resources of all physical machines to provision a large number of virtual machines. An example of SaaS which is the main subject of this thesis is the Content Delivery Network (CDN). Indeed nowadays, Internet traffic is dominated by data distributed. Content distributed over the top creates a bi-directional dependence between CDN and network providers: CDN customers, and, in turn, end users, depend on the underlying network infrastructure, and user experience is affected by its conditions. Network operators, on the other hand, are vulnerable to the traffic dynamics caused by time-varying shifts in content demand. At the same time, network providers wish to take advantage of their regional presence and proximity to end users to enter the content delivery market.

This has led to the emergence of telco CDNs, i.e., content delivery infrastructures owned and operated by network providers. In this case, the network operator installs data centers at its points of presence (or other points strategically located in its network) and offers a CDN service advertising high-performance delivery due to user proximity. From the perspective of a content provider, traditional and telco CDNs are not competitive, but, rather, complementary services: A telco has the advantage of proximity to end users, but typically this is limited to specific geographic locations where it has presence. On the contrary, traditional CDN players compete offering a global service.

In this chapter, we introduce the context of this thesis and we review the literature in CDNs architectures. Furthermore, we define the NFV paradigm which is one of the key enablers of virtualized CDN addressed in this thesis. Finally, we introduce the MEC in NFV architecture, a new line of research followed during the thesis.

2.2 Content Delivery Network (CDN)

Nowadays technologies evolve with the need to deliver diverse content on a minimum time and at high speed. CDN from its first appearance in [22] as a solution in the processing of the large content (media, audio and video) on the web and until now is designed to meet those challenges. Both the distribution of media resources and the minimization of the service deployment are important for the content provider, the service operator and the end-user. Designing a CDN architecture leads to have an overview of the end-users and ensure that they receive an optimized Content Delivery. With CDNs, contents are distributed to cache servers close to users. In particular, CDNs have various Points of Presence (PoP) with cache (surrogate) servers (close to the end-user) that store copies of the original content [85]. FIGURE 2.1 illustrates the basic overview of the CDN. As can be seen from this figure a CDN replicates content from the original server to cache servers in different locations all over the world in order to deliver content to the end-users (refer as clients in the figure) within a reasonable time.

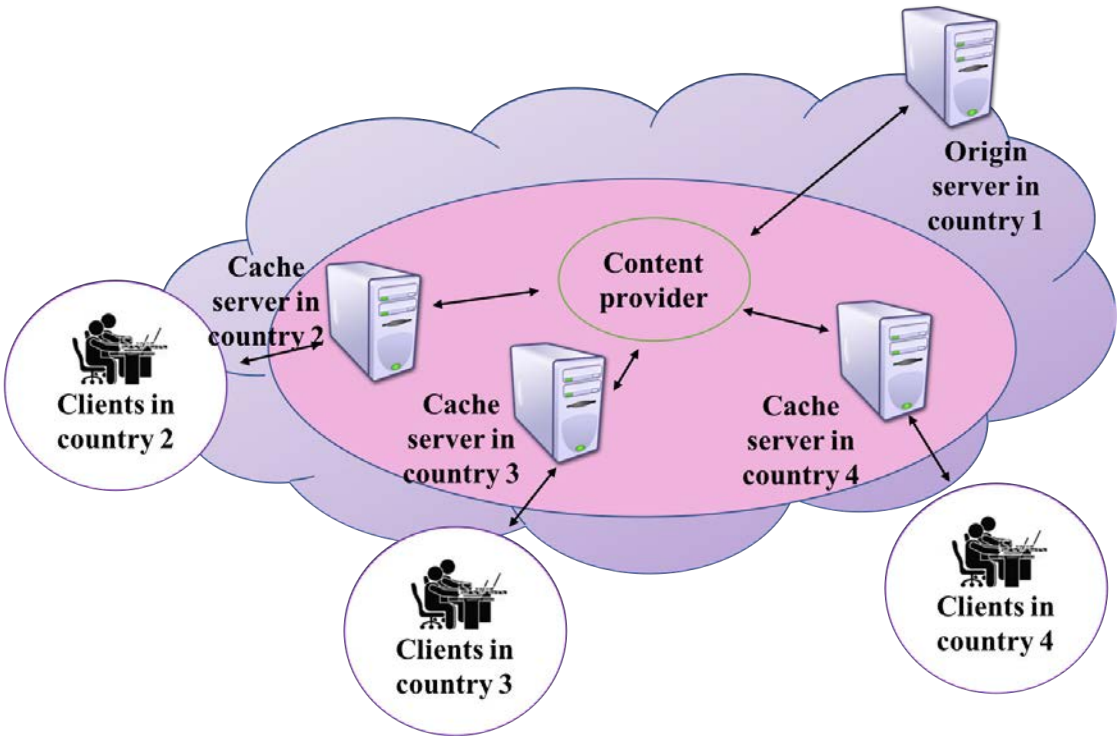


FIGURE 2.1 : Content Delivery Network overview.

Few works presented the layered architecture of CDNs [86, 3]. The idea is to highlight the main protocols used in CDNs in a layered manner. According to [86] the main layers are: (i) Basic Fabric layer which is the lowest layer of a CDN; it provides the infrastructural resources such as networking infrastructure, Symmetric multiprocessing (SMP), file servers, etc. All these components are running different software systems, i.e, operating systems. (ii)

Communication and connectivity layer provides CDN Internet protocols in addition to the classical internet protocols (TCP/UDP, FTP). The CDN protocols used for communication, caching and delivery of a content are: Internet Cache Protocol (ICP), Hypertext Caching protocols (HTCP) and Public Key Infrastructures (PKI) respectively. (iii) CDN layer is the key component of the layered architecture, providing CDN services, CDN types and different types of content such as text, images, audio, video, etc, to the user. (vi) End-users layer is at the top of the layered system which consists of a local interface where the end-user can request a CDN by accessing the content provider's web site. FIGURE 2.2 represents the layered architecture of a CDN.

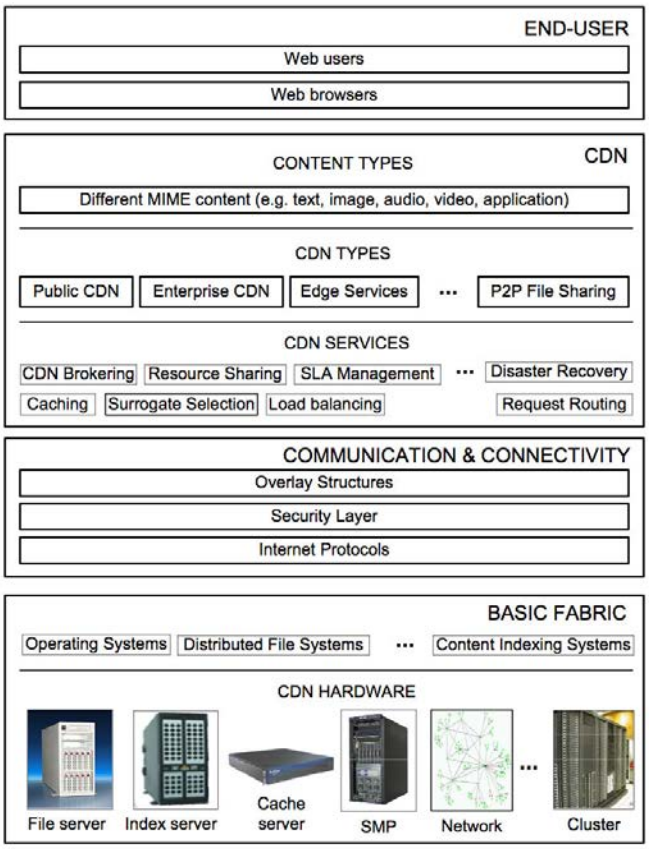


FIGURE 2.2 : Layered architecture of a CDN [86]

The European Telecommunications Standards Institute (ETSI) proposes a standard CDN functional architecture [25]. The high level components architecture is presented in FIGURE 2.3. Each component has its own features, which are:

- The Content Deployment component generates copies of the content within the CDN. The content deployment is also in charge of the delivery, storage resources and the establishment of the optimal deployment policy of the replica servers in the deployment of a content.

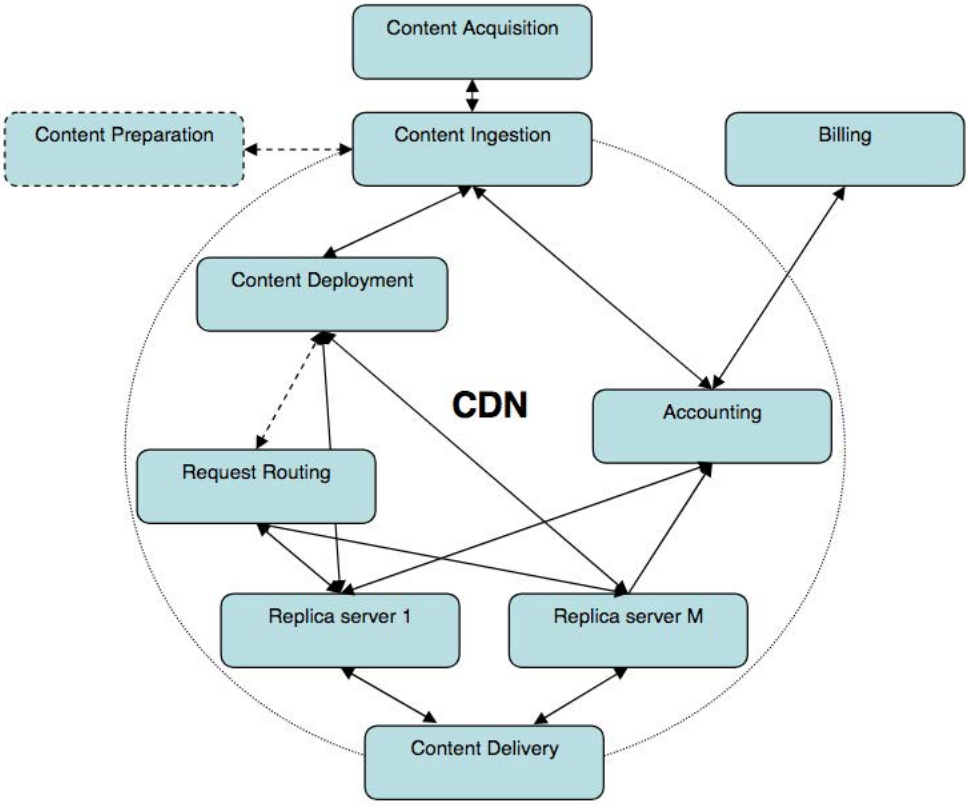


FIGURE 2.3 : Architectural components of a CDN [25]

- The Request Routing component finds the appropriate replica servers for the client request for content. It keeps an overview of the content stored in the CDN.
- The content Delivery component is the main contact with the end user; it delivers the content.
- The Accounting component records the usage of the CDN, which can be used as a traffic monitoring.
- The Billing component uses information from the Accounting one, for example in order to report the state of the traffic.
- The Content Preparation component is responsible for the transcoding, encryption or other functions of this type.
- The Content Ingestion component prepares the content for acquisition.

Other organizations of standardization such as IETF [87] propose a CDN Interconnection (CDNI) where the objective is to model the communication between administered CDNs in order to increase their outreach. The CDNI standard allows an authoritative upstream CDN to use the infrastructure of an autonomous CDN that can offer the best content delivery

service. This communication includes the addition of a content, its modification (update, delete, convert, adapt, etc.) between multiple CDNs.

From the industrial side, the commercial provider Akamai is considered as a global leader in the CDN market [82]. Nowadays Akamai delivers up to 30% of the world's web traffic. The followed approach aims to reach all edges of the Internet by deploying data centers in a large number of end-user ISPs [82]. The main components of Akamai CDN's architecture are:

- The mapping system translates the domain name typed by the user into the IP address of the closest edge server.
- The edge server as part of the edge server platform handles the request received from nearby users and serves the requested content.
- The edge server platform is a large deployment of multiple servers located all around the world.
- The transport system is responsible of delivering the requested content from the origin server to the edge server.
- The communications and control system distribute control messages and status information such as updates when needed.
- The data collection and analysis system gather information such as network data.
- The management portal provides information on the way that the content and applications are served to the end users. It also provides reports on audience demographics and traffic metric

FIGURE 2.4 illustrates the mentioned components of the CDN architecture of Akamai [82].

The collaboration between CDN providers and ISPs is the key to provide contents in good quality. In this spirit Akamai proposes *Aura Licensed CDN [1]*, which consists of offering their CDN services to their customers. The advantage is that the operator (ISP) can own and operate the CDN. Another type of ISP-CDN collaboration is the Telco CDN where the Operators develop their own CDN infrastructure. The Telco CDN is the subject of this thesis which we detail in section 3.2. To empower their role in the content delivery value chain our position is that the Telco CDN can open up to take advantage on the virtualization and cloud management framework in order to lease infrastructures on demand in a flexible and dynamic manner. One advantage to content delivery is its inherent network awareness, and the proximity to end-users. This can be exploited to improve user experience while saving on cost. Our positon is more deeply detailed in the next chapter.

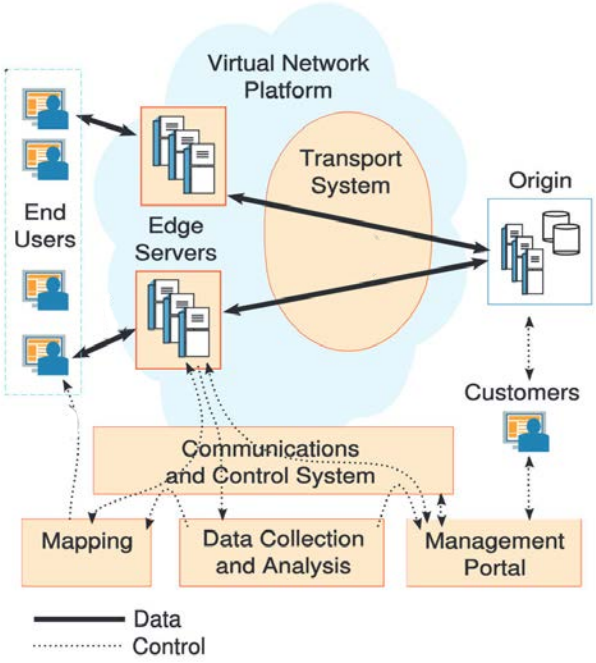


FIGURE 2.4 : System components of a delivery network of the CDN provider Akamai [82]

As mentioned previously, we believe that the Telco CDN has to exploit the virtualization and virtual CDN. This is where NFV comes in. NFV is a group of multiple networks which can operate on a given physical network [14]. Given its big impact in both industrial and research bodies and its importance in the achievement of this thesis we detail in the next Section this new paradigm and we present implementations of NFV platforms from the state of the art.

2.3 Network Functions Virtualization (NFV)

Network Functions Virtualization was introduced in 2012 by ETSI [24] in a consortium of ISPs. The goal was to specify requirements for the deployment of Network Functions as software on Commercial Off-The-Shelf (COTS) hardware. One of the main objectives was to reduce cost which leads to multiple approaches and propositions. The ETSI describes a high-level function architectural framework and design of NFV [80], depicted in FIGURE 2.5:

The main components of the ETSI NFV architecture [80, 70] are:

- The Network Function Virtualization Infrastructure (NFVI) block includes the software and the hardware resources needed to run the VNFs. The shared physical resources are the computing network, storage and hardware.The virtual layer is the abstraction of the shared physical resources. In practice, these virtual resources are represented in one or multiple Virtual Machines (VMs), except for the virtual network which is

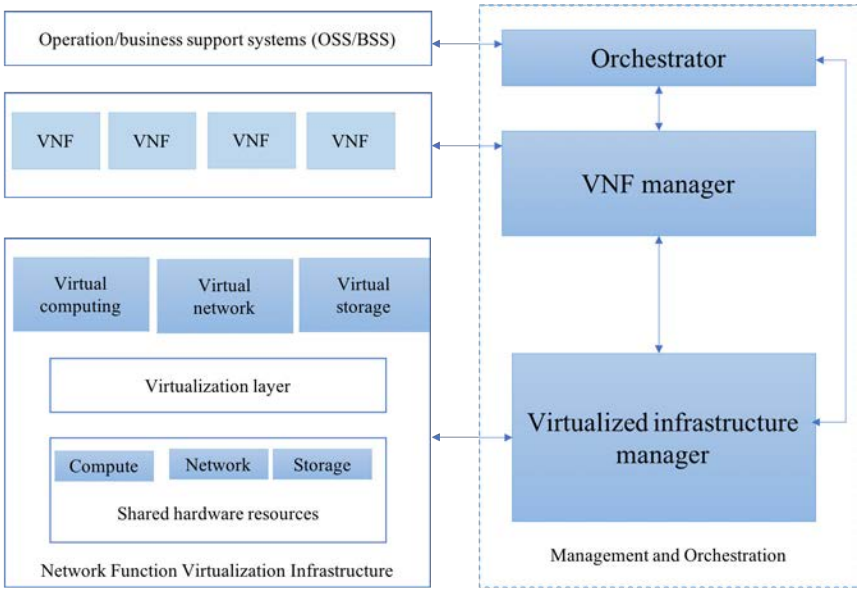


FIGURE 2.5 : Functional blocks of NFV architecture [80]

represented as virtual links and nodes.

- A VNF is an implementation of a network function and can include one or more internal components. This make a deployment of a VNF over one or multiple VMs.

- The Virtualized Infrastructure Manager (VIM) as its name indicates, is responsible for the control and the management of the interaction of a VNF with the software resources (computing, storage and network) as well as their virtualization. It is also responsible for the resource allocation when requested by the VNF manager.

- The orchestrator or NFV orchestrator (NFVO) is responsible for the orchestration and the management of the NFVI and software resources. The NFVO has a global view of the system.

- Operation Support System (OSS) and Business Support System (BSS) are external entities representing the service provider. They include the collection of systems and management applications that service providers use to operate their business.

After the NFV architectural framework, ETSI proposes a reference architecture of management and orchestration for the provisioning of NFV called *NFV-MANO* [77]. In particular NFV-MANO consists of the three main blocks represented on the right in FIGURE 2.5.

An example of NFV-MANO implementations is the cloudNFV project [1] which is a platform for implementing NFV based computing and Software Defined Networks (SDN). In the same spirit an effort in VNF-as-a-Service worth noticing is T-NOVA [27, 105], an EU-funded FP7 project aiming to offer a marketplace for VNFs, where VNF providers will be making available their functions to be deployed over the infrastructure of a network or cloud service provider, developing the necessary support for VNF brokering, management and service delivery. Our architecture is in line with the NFV MANO reference architecture and it is detailed in the next chapter.

2.4 MEC in NFV environment

Another line of research explored in this thesis is Mobile (or Multi-access) Edge computing (MEC), which is a key component in future 5G mobile communication systems. In particular, MEC is built on recent advances in mobile cloud computing [61]. MEC comes with the development of mobile devices as a solution for the intensive computation and delay sensitivity of mobile applications [64]. The ETSI [73] refers to MEC as a system where computing resources are installed within the Radio Access network (RAN) [2] close to mobile devices. ETSI has also published documents that (i) define interfaces to interconnect MEC applications with the mobile network data plane to interact with the user data, (ii) specify interfaces to run MEC applications within a virtualization platform, and (iii) provide MEC services to leverage the mobile operator, by exposing low-level RAN information to third-party application developers, allowing the development of context-aware services. MEC is more mature and ready to be deployed compared to FoG, where the first release of the architecture has been recently issued,[3] and mainly targets IoT services. FIGURE 2.6 illustrates the MEC system. It can be seen that the computing equipment, *the mobile Edge host (ME host)*, is installed close to the base stations (BS). The difference between Edge and Central cloud is that in the former the computing is managed locally by the network operator. This can reduce access latency to the service/application.

Edge computing is becoming a key pillar of 5G, especially because of its low latency during a service deployment. The fact that it is at the edge close to users helps to satisfy the user request in a minimum delay, while longer delays are noticed in cloud computing, where datacenters are not necessarily close to the user. However in terms of resources

1. http://www.cloudnfv.com/WhitePaper.pdf
2. In a RAN, radio sites provide radio access and coordinate management of resources across the radio sites. A device is wirelessly connected to the core network, and the RAN transmits its signal to various wireless endpoints, and the signal travels with other networks' traffic: https://www.sdxcentral.com/5g/definitions/radio-access-network/
3. https://www.openfogconsortium.org/ra/

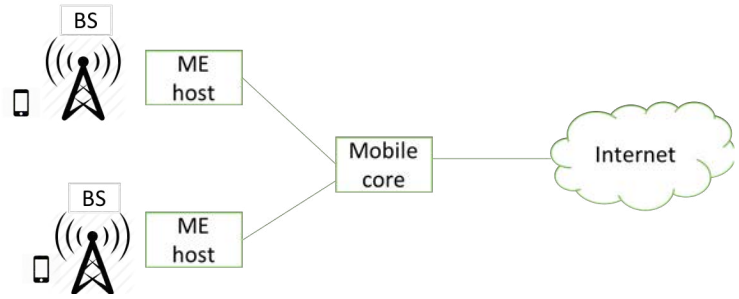


FIGURE 2.6 : MEC system

the traditional cloud is better supplied. This brings ETSI [72] to investigate a new line of work namely MEC in NFV architecture. Bringing both technologies together (MEC-NFV), by integrating the edge within the traditional cloud will lead to create a set of APIs within small datacenters at the edge.

The ETSI reference architecture of MEC-in NFV is illustrated in FIGURE 2.7. This architecture allows an easy integration of MEC elements such as Mobile Edge platform (MEP), Mobile Edge Orchestrator (MEO) and Mobile Edge platform Management (MEPM) within the NFV environment [72]. As can be seen there are common components with the NFV reference architecture presented in the previous section.

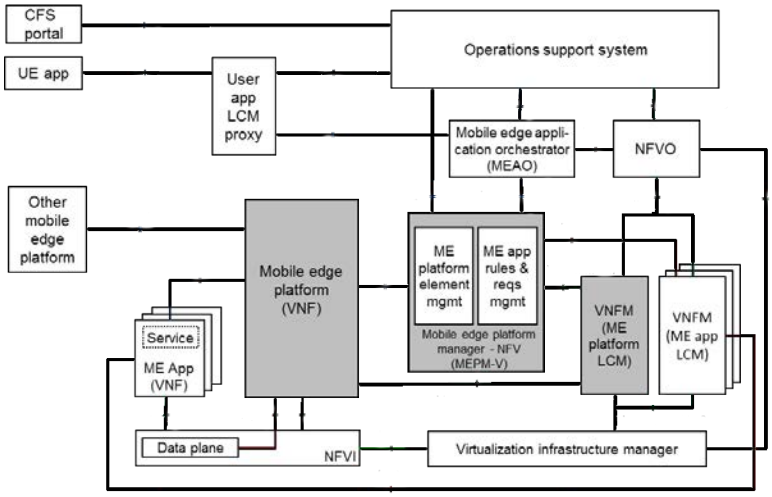


FIGURE 2.7 : MEC in NFV reference architecture [72]

We summarize the main components of the ETSI MEC reference architecture in an ETSI NFV environment in the following [72]:

- *NFV Orchestrator (NFVO)*: the role of the NFVO is to instantiate the virtual resources for the MEC applications as well as for the MEP.

- *Mobile Edge Platform (MEP)*: in this architecture the MEP is run as a VNF where it stores service instance templates and provides them to the NFVO to take deployment decision.

- *Mobile Edge Platform Manager (MEPM)*: is responsible for the ME platform element management, the lifecycle management of the ME application and the host level policies management. It also maintains information on available ME services in the ME system. It also interacts with the OSS for fault configuration and performance management

- *Mobile Edge Application Orchestrator (MEAO)*: this component has the view of the whole ME system, it maintains the information about all the deployed ME hosts, the services, the available resources in each host, the ME applications that are instantiated and the topology of the network. The MEAO is also responsible for installing the ME applications in the system, checking their integrity, authentication and validation. More specifically, the MEAO is in charge of the placement of MEC applications,and their instantiation using the NFVO.

In this context, Sciancalepore et al. [95] designed a double-tier MEC-NFV architecture. Their approach is aligned with the ETSI NFV MANO specifications previously presented. The proposed scheme introduces a new management subsystem in order to provide application-oriented orchestration capabilities which are offered to the mobile edge applications and their management. Our vision on MEC is presented in chapter 5 where we focus on a placement scheme that takes into account edge cloud in an NFV environment, and consider specific criteria such as latency, in view of supporting uRLLC.

2.5 Conclusion

To better understand the context of this thesis, we introduced in this chapter two main subjects, namely CDNs and NFV. We defined the traditional CDN and its derivatives such as managed CDN and the telco CDN. We also presented multiple architectures of a CDN where we defined each key components. Then we described the expansion of NFV as a key technology in content/service delivery and we highlighted the key architectures of NFV the *NFV-MANO*. Finally, we explored the state of the art on MEC in NFV environment. All these challenges inspire us to propose in the next chapter, a design and implementation of an architecture for on-demand deployment of a vCDN infrastructure over a telco cloud. In particular, given that the service is provided over a virtualization platform, we carry out an extensive measurement campaign where we evaluate the performance of key enabling

technologies for our vision (in our case, virtualization based on kvm and containerization based on docker).

CHAPTER 3

AN ARCHITECTURE FOR ON-DEMAND SERVICE DEPLOYMENT AND PROVISION OVER A TELCO CDN

3.1 Introduction

Internet traffic is dominated by data distributed over Content Delivery Network (CDN) infrastructures, and the current Internet ecosystem is, to a significant extent, shaped by the interactions of different key actors with often conflicting objectives, including Over-the-Top (OTT) content providers, IP transit providers, CDN infrastructure providers and ISPs [63].

In this chapter, with cloud orchestration frameworks and NFV as our enabling technologies, we present our design. Our scheme offers the flexibility to a telecom operator to lease its CDN infrastructure in a dynamic manner, providing a virtual CDN (vCDN) service that can be deployed on demand over the operator's private cloud. This can be considered as an evolution towards opening a telco CDN to potentially (but not exclusively) smaller-scale content providers. Our basic design goals are (i) to offer a well-specified, extensible northbound interface to customers, which will allow them to express service demand specifications and performance constraints, ensuring them sufficient control at the service level, but abstracting internal network and infrastructure details, and (ii) to be able to combine customer-provided demand dimensioning information (e.g., target number of users/video streams per region) with network and compute infrastructure awareness for optimal resource allocation. For the latter, we argue for a measurement driven approach: It is important to know the capabilities of the underlying technologies and be aware of the relationship between service workload and user experience in order to take informed resource allocation, placement, and real-time management decisions, maintaining the level of service agreed with the customer while minimizing operational costs; the experimental results we present in this work serve to this end. Our work thus aims to provide the technical elements to build mechanisms for cost- and quality-optimized CDNaaS provision. These elements consist in the architecture support to develop algorithms and mechanisms for service delivery (resource allocation, service place-

ment, and elastic resource/service management included) and in the necessary quantitative insight to guide them. We consider these mechanisms, however, as a separate line of research which deserves attention in its own right and focus on them in chapter 4.

This chapter is organized as follows: In Section 3.2 we overview the state of the art in Telco cloud, data center virtualization and NFV. In Section 3.3 we present our design and implementation of a cloud-based architecture for the provision of a virtual CDN service tailored to telecom providers. We also present results of testbed experiments from our prototype implementation on top of OpenStack [83]. In Section 3.4 we carry out an extensive experimental campaign to explore the capabilities and limitations of the virtualization technologies we apply, quantifying the relationship between workload and performance from a system operator and a user perspective.

3.2 State of the art

3.2.1 ISP-CDN collaboration and Telco CDN

Telco Content Delivery Network (Telco CDN) is born due to the exploding high definition video streaming traffic. It provides services which save on traffic toward the Internet [60]. Telco CDN involves the installation and management of CDN servers by the Internet Service Providers (ISPs) within their network. Telco CDN has been created for network operators to optimize resources for content delivery, in an Internet Service Provider (ISP) managed CDN infrastructure [60].

In typical CDNs, content delivery functionality is replicated across a number of servers, potentially at a global scale. Content providers delegate the process of delivering their content to CDNs, and the latter select the most appropriate servers to respond to user requests for content items, balancing among maximizing user experience, minimizing delivery cost, and optimally managing CDN server load. CDN performance critically depends on the conditions in the underlying network path between the user and the content location, which however, being managed by network providers, is outside the control of the CDN. Without CDN-ISP collaboration, the CDN relies on estimates which are often not accurate. On the other hand, this network-unaware server selection process can cause unpredictable traffic shifts which can significantly impact the operation of the ISP network. Therefore, both ISPs and CDNs face a fundamental challenge, which is analyzed by Frank et al. [33] along with a thorough description of the technical issues and typical architectures for content delivery over the Internet.

Incentives for collaboration between ISPs and CDNs thus emerge and different strategies can be followed, from a non-cooperative case, where the CDN operates over the top in a network-agnostic manner, to the case where the ISP deploys and controls its own CDN infrastructure, which is the target environment for our work. This can happen either by developing its own content delivery solution, or by acquiring a license to CDN software (licensed CDN), potentially sharing revenue with the CDN provider. For example, the architecture that we propose implies a telecom-operated CDN and many of the technical and managerial challenges stemming from the need for ISP-CDN collaboration are alleviated, since the whole infrastructure is administered by a single entity. Other options involve CDN operators directly installing their servers inside the ISP network and/or setting up specific interconnection agreements.

Poese et al. [89] elaborate on mechanisms for the ISP to offer assistance to the CDN, without it being necessary for each party to reveal sensitive details about their internal workings. They propose enhancements to the ISP's DNS infrastructure in order to direct user requests to the most appropriate server, taking into account optimization criteria set by the ISP and aiming to improve user experience. In this context, NetPaaS [34] provides the interfaces and the management tools to CDN operators to deploy their service in an ISP-assisted manner, at the same time offering informed user request to content server assignment. It operates for both physical CDN deployments (bare metal servers placed by the CDN in the ISP network) and virtual ones, if they are supported. Our approach assumes a different ownership model, i.e., the ISP is in full charge of the underlying infrastructure and the CDN service. Second, in our case, the customer is the content provider which requests the deployment of a CDN service in a SaaS manner, while for NetPaaS the service offering is at the infrastructure or the platform level and the CDN is in full control of the virtual resources. Third, the resource allocation decision in our case is carried out fully by the service operator (ISP) based on measurement-driven empirical models. We should further note that our focus is more on the infrastructure support for CDNaaS provision, including an NFV-MANO-compatible architecture design (detailed in the next Section), than on CDN-service-level details. As such, our system can support various CDN-level functionalities and can further expand towards supporting licensed CDN cases, where the service operator deploys and controls the licensed CDN software as a set of VNFs.

Herbaut et al. [42] analyzed the ISP-CDN collaboration models by using game-theoretic tools, identifying that this collaboration can be implemented on top of an NFV infrastructure and can be mutually beneficial under certain circumstances. In that model, the NVF platform is hosted by the ISP and forms a marketplace where CDN operators can upload VNFs, which can in turn be chained by the ISP into different types of CDN services and under

different pricing options. Business-wise, our approach (presented in Section 3.3) is more on the telco CDN side, and, thus, more ISP-centric: The network provider fully controls both the NFV/cloud platform and the content delivery service. Technology-wise, although both approaches share some common ground, we focus more on the technical aspects of such a scheme, providing a detailed architecture design and its implementation.The NF-as-a-Service approach of T-NOVA project mentioned in chapter 2 shares similarities with our work, but the overall target application and business environment is different. Our architecture is designed and optimized for content delivery. Resource management and other relevant low-level deployment decisions are taken by the operator, based on customer demand and constraints, and accurate knowledge of the structure and conditions of its infrastructure.

A driving force behind the deployment of telecom-operator-owned content delivery infrastructures is the fact that telcos aim at empowering their role in the content delivery value chain by exploiting their key advantages of network control and end-user proximity, a role that is traditionally challenged by over-the-top content delivery without their strong involvement. The economics of such vertically integrated CDNs are studied by Maillé et al. [65], who make some interesting observations on the incentives of an ISP to operate its own CDN: In the face of competition with other ISPs not operating a CDN, and depending on the price sensitivity of users, while from a user experience and a regulator perspective a vertically integrated CDN is always beneficial, for their specific model settings there are circumstances under which the ISP has incentives not to run its own CDN. This implies that the decision of a network operator to run a CDN should be carefully studied. Although there is generally a clear business case for ISP-CDN integration, as also demonstrated by network providers rolling out their own CDN infrastructures, a study of such incentives and economic interactions between the involved actors is outside the scope of this thesis.

Despite some key advantages with respect to network awareness, which can improve user experience and thus, potentially, increase the customer base, telco CDNs are limited by their regional coverage. A strategy to overcome this limitation is CDN federation, which is also a subject of the IETF CDN Interconnection (CDNI) Working Group [17]. Given the complementary competitive advantages of telco and traditional CDNs, Lee et al. [56] perform a game-theoretic analysis of the strategic interactions between the two types of players. Importantly, they study the conditions that can lead to alliances among telco CDNs and provide evidence that if a telco CDN properly manages to offer better service quality exploiting its competitive advantages (e.g., joint traffic engineering and content distribution), market benefits are possible. Telco CDN federation can take various forms. As Lee et al. show [56], there are cases under which the potential for full resource pooling and revenue sharing among

the federation is beneficial, although in most cases resource pooling on its own brings more benefits to each individual telecom operator. Regarding critical decisions that a telco CDN operator needs to take, Kamiyama et al. [51] study and identify these decisions such as the regional placement of data centers, and where and how to cache content.

Spagna et al. [99] propose an overview of some challenges, design goals and principles for a telco CDN.

3.2.2 Data center virtualization

Our work is conceptually related with the Virtualized Data Center (VCD) abstraction [39]. In both cases, following a customer request, a set of cloud resources are allocated, operating in isolation on a shared infrastructure, potentially scaling dynamically. However, the VCD abstraction is closer to the Infrastructure-as-a-Service (IaaS) model, where the customer leases resources in the form of VMs with specific compute, storage and bandwidth guarantees, appearing as a dedicated (but virtual) data center for the execution of arbitrary platforms and applications. The vCDN mostly follows the SaaS model: The customer (content provider) requests for the instantiation of a virtual CDN infrastructure on top of a telco cloud for the diffusion of its content. As such, the infrastructure- and platform-level functionality is managed by the service operator. The content provider is agnostic to the infrastructure and potentially even the platform being used. We should note that at the data center network level, the necessary network virtualization functionality needs to be in place in order to offer the appropriate traffic isolation and network resource provisioning in a naturally multi-tenant environment.

3.2.3 Network Functions Virtualization (NFV)

NFV has been proposed in order to take advantage of virtualized technologies to offer a way to design, deploy and manage networking services [70], and is becoming a key technology for future large-scale service delivery [49]. NFV involves carrying out in software networking tasks that were traditionally performed by costly, special-purpose hardware. It is facilitated by developments in cloud computing and networking technologies, which have helped run such functionality on top of commodity hardware, offering on-demand scaling and automatic reconfiguration capabilities, decoupling the necessary service logic from the underlying hardware [41], and at the same time allowing (i) network operators to deploy and manage services with more flexibility and reduced capital and operational expenses, including space and energy costs, (ii) third-party application/service providers to innovate the market, lowering the technological barriers stemming from compatibility issues, and (iii) telecom equipment vendors to focus on the functionality of their solutions and to expand

their service portfolio. In NFV, services can be decomposed into a set of VNFs which are implemented in software running on one or more physical servers [70].

Mijumbi et al. [70] highlight the limitations of NFV and its relationship with the Software Defined Networking (SDN) and cloud computing. The authors survey the state-of-the-art in NFV, focusing on different challenges such as operating and capital expenses (OPEX[1] and CAPEX[2]).

NFV is being applied to a diverse set of functions. In our architecture (next Section), the basic components such as service orchestration, virtual infrastructure management, etc. and the components of the CDN service (caches, load balancers, name servers, etc.) are implemented as VNFs.

In another context, Yousaf et al. [110] propose SoftEPC, a design which takes advantage of cloud computing technologies to virtualize the Evolved Packet Core (EPC) of the 3GPP LTE/EPS mobile cellular network architecture, so that the network can respond more efficiently to dynamic traffic demands. Taleb et al. [100] provide an overview of the challenges and design alternatives to achieve this EPC-as-a-Service vision.

It should be noted that intense NFV standardization efforts are currently underway. As mentioned in the previous chapter, the ETSI has specified a Management and Orchestration framework for NFV (NFV-MANO [77]), and one of the proposed NFV use cases is the provision of virtualized CDN services [79, Use Case #8]. We put our design in the context of NFV-MANO in Section 3.3.5.

One of the challenges in NFV is appropriate function placement. Clayman et al. [16] propose a VNF management and orchestration framework and experiment with different virtual router placement algorithms. In a similar spirit, Moens and De Turck [75] present a theoretical model for VNF resource allocation, focusing on an environment where a base load is handled by physical hardware, and virtual resources are used on demand to deal with load bursts.

1. OPerating EXpenditure: the cost needed to operate the network, i.e., site rental, leased line, electricity, operation and maintenance and upgrade [13].
2. CAPital EXpenditure: the expenditure related to network construction which covers the network planning to site acquisition, RF hardware, baseband hardware, software licenses, leased line connections, installation, civil cost and site support such as power and cooling [13].

Network functions are traditionally subordinated in specific hardware which makes it is difficult to add a new service because of high cost and system stability. These issues have been addressed by proposing to place virtualized services on demand in Cloud data centers. As pointed out by Wood et al. [104], the flexibility offered by NFV comes with a performance cost due to virtualization. They therefore identify the need for a carefully designed NFV platform, coupled with a sophisticated SDN control plane. Our work addresses such challenges, by experimentally quantifying the performance capabilities of core enabling virtualization technologies, and by offering expressive management and control interfaces to customers and flexible service representations, which enable to focus on optimal virtual service construction considering customer and operator cost, performance, and quality constraints.

3.3 A Framework for CDNaaS provision

3.3.1 Features

We present an architecture which allows a network operator to virtualize its CDN infrastructure and lease it to content providers on demand. Our approach can be viewed as an evolution of the telco CDN model, offering the operator the flexibility to provide lower-cost CDN solutions. These could appear more attractive to small-to-medium content providers, but other options are possible. For instance, the virtual CDN instantiated by a customer can be coupled with other CDN infrastructures; the flexibility offered by our design allows the customer to, e.g., use the leased infrastructure to respond to predicted traffic surges at specific regions, taking advantage of the network operator's regional presence. From the perspective of the CDNaaS provider, our design allows for more efficient use of its infrastructure resources, compared to a less dynamic resource reservation model with static allocation of data center resources to customers. Our approach can also operate transparently over federated clouds, which can be a strategy that telecom operators may follow to address potential limitations due to restricted geographical coverage.

3.3.2 Architecture

Our design involves various functional blocks, communicating via well-specified interfaces. This decouples their operation from any physical location, allowing the CDNaaS provider to execute any of these blocks autonomously as virtual functions over its own (or any) cloud infrastructure. Our layered and modular design also aims to abstract the details of the underlying cloud infrastructure and thus avoid lock-in to a specific cloud management platform. One of our core design principles is to expose open APIs to customers, but also among the

components of our architecture, and to design with extensibility in mind, so that our scheme can be extended towards a generic Any-as-a-Service model. In this Section we provide the main components of our CDNaaS architecture (figure 3.1), their functionality and their interactions.

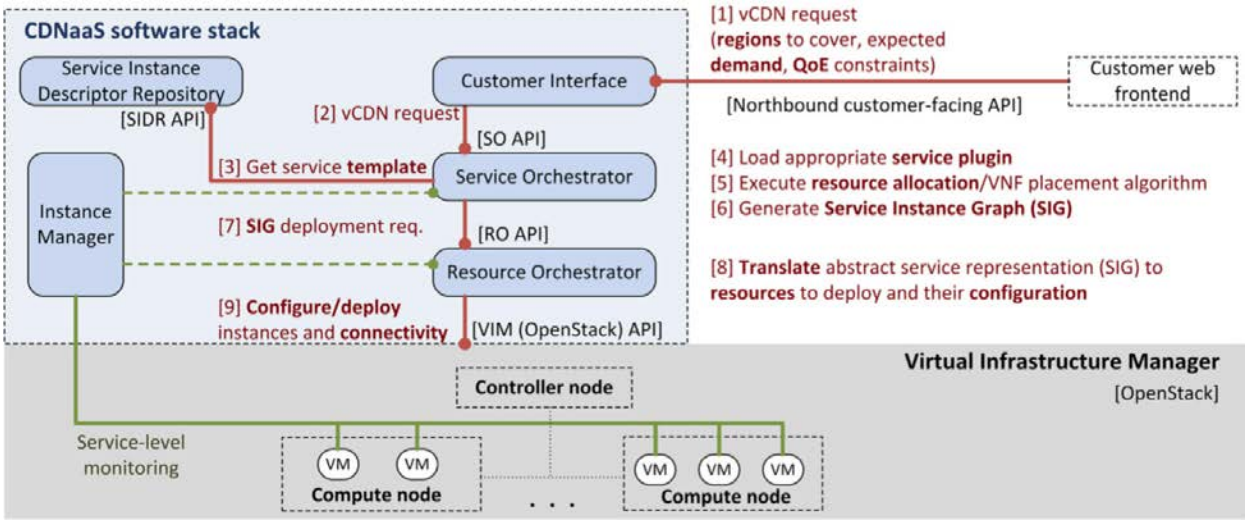


FIGURE 3.1 : CDNaaS architectural components [31].

A Customer Interface Manager

Our system provides a RESTful northbound API, through which customers can request to deploy a vCDN over the telco cloud. This API exposes information on the supported vCDN service types and the regions the operator has presence, abstracting information about the underlying network and cloud infrastructure, and mediating the communication between customers and the Service Orchestrator (SO). Using the northbound API, a customer can specify its service requirements per region that it wishes to cover, and in particular (i) demand specifications, i.e., how many clients it wishes to serve per region, and what is the service lease duration, and (ii) quality specifications, which can be considered as specific Service-Level Objectives (SLOs), such as target QoE ratings, desired response times, target availability levels, etc. After a service has been successfully deployed, this API returns an entry point to the vCDN instance so that the customer can manage its deployment (e.g., infuse content in the vCDN).

B Service Instance Description Repository (SIDR)

Each service supported by our scheme has some inherent requirements and constraints. These are encoded in a service instance template, which provides information such as the

minimum number of VNF instances of a specific type that need to be deployed for a vCDN service instance, constraints with respect to the processing and memory resources that should be available to each VM, etc. Although we mainly focus on the provision of a virtualized CDN service, our design includes an extensible service template description language which can support further services. The SIDR component stores these service instance templates and provides them to the Service Orchestrator to drive service deployment decisions.

C Service Orchestrator (SO)

The Service Orchestrator coordinates vCDN service deployment. After receiving a request from the customer, it derives an appropriate resource allocation and VNF placement, taking into consideration (i) the inherent service requirements, as expressed in the service instance template, (ii) the client demand and quality specifications, included in the service request, and (iii) its operational capacity. Depending on the specific vCDN flavor selected by the customer, the appropriate algorithms are executed, resources are reserved, and the instantiation and termination of the vCDN are scheduled for the desired times, as requested by the customer. The result of the execution of these algorithms is a Service Instance Graph (SIG), which maps VNF instances (VMs) to physical nodes. The SIG is then passed on to the Resource Orchestrator (RO) for deployment.

D Resource Orchestrator (RO)

The RO component is responsible for the deployment of a service instance on the underlying telco cloud. The cloud infrastructure is managed by the VIM component, in the ETSI NFV-MANO [77] terminology. In our implementation we are using OpenStack, the de facto VIM platform. The RO receives the SIG that the SO derives following a customer request, and uses the northbound VIM API to launch and configure VNF instances. Note that the RO takes care of specific technical details, such as IP address management for the deployed instances, but also implements the necessary service-level logic (e.g., the boot-time configuration of DNS servers to appropriately handle geolocation or the setup of cache virtual instances to proxy user requests to origin servers); this logic may differ across the various vCDN flavors available from the operator. Since the RO is the only component which directly interacts with the telco cloud, the other components of our design are agnostic to low level platform interfaces and technical details. Changes at the infrastructure level need only be reflected at the RO.

E Instance Manager (IM)

There are specific management tasks which are common across any vCDN flavor, such as vCDN instantiation and termination. However, more elaborate critical functions need to be performed in a different way for different vCDN types. This functionality is carried out by the IM and pertains to both the SO and the RO layers. For example, complex dynamic resource management algorithms can be implemented here, which use the SO and RO APIs to scale up or scale down a vCDN service instance when deemed appropriate. Such functionality requires real-time service-level monitoring, which is also performed by the IM.

F vCDN service-level components and life cycle

In our reference vCDN deployment scenario, a video content provider operates its origin server(s) in its premises or in external public or private clouds, where it places original video content. Then, it uses the customer-facing API to request the instantiation of a vCDN covering specific regions where the operator has presence, to serve a target maximum number of parallel video streams (demand specification) with the desirable video QoE (quality specification) for a given period. The vCDN instance that will be created will include a number of caches distributed across the operator's regional data centers. The vCDN service supports a two-level load balancing: A user's DNS request for the URL of a content item is resolved to the appropriate regional data center, based on the user's geographic location, as inferred from the latter's IP address (DNS geolocation). Then, HTTP requests are balancing across all cache VNF instances deployed in a regional data center. Upon receiving the customer request, the SO runs an algorithm to calculate (and reserve) the resources necessary and a VNF instance placement on the appropriate regional data centers in the form of a SIG, and uses the RO API to request its deployment on the provider's infrastructure. The RO, which is responsible for translating the abstract service representation (SIG) to an actual deployment, automatically configures one or more DNS VNF instances for request geolocation, configures cache VNF instances so that they proxy all user requests towards the content origin server(s), and configures each region's load balancers. Eventually, the customer entry point, with specific information on the vCDN instance (such as the IP address of the DNS server) is included in the API response. During the operation of the vCDN instance, it is monitored at the service level and reconfigured if necessary (e.g., scaling up the allocated CPU resources to cope with increased demand to maintain the desired QoE level) by the IM. The vCDN service is terminated and the respective resources are released either automatically, when the lease specified in the customer request expires, or when the customer explicitly requests its termination using the northbound API.

3.3.3 Implementation

A Technologies

As a proof of concept, we have implemented our CDNaaS architecture on top of OpenStack. VNF instances (caches, DNS servers, etc.) are executed as Debian Linux virtual machines on kvm [23] hypervisors (compute nodes, in the OpenStack terminology). We are using nginx [24] to implement HTTP server/caching functionality, a choice motivated by its wide adoption. The components of our scheme (customer interface, SIDR, SO, RO) have been implemented in Python and communicate over HTTP by exchanging JSON-encoded information. Service templates and service instance graphs are also representd as JSON objects. Our RESTful northbound API has also made it straightforward to create a web-based customer front end.

B Service plugins

To deal with the different requirements of the various vCDN versions offered by the operator and to ensure the system's extensibility, we introduced the notion of the service plugin. Our software architecture provides a plugin interface which exposes hooks for its main components, where the funtionality pertinent to each one of them is to be implemented. In particular, for each vCDN type, a service plugin needs to be developed, with the following functionality per component:

- **Service Orchestrator:** The resource allocation and VNF placement algorithm for the specific vCDN type, the output of which is a SIG, is specified here. The SO plugin functionality is also responsible for reserving the resources that will be needed for the vCDN service instance, if these are available.

- **Resource Orchestrator:** Using the RO API, the SIG is deployed over the underlying cloud infrastructure. The plugin is responsible for the service-level functionality, i.e., appropriately configuring VNF instances at boot time.

- **Instance Manager:** After successful service deployment, the SO starts an instance management function, which carries out the appropriate monitoring tasks and executes dynamic resource management algorithms. Elasticity functions are implemented here. For example, for a video streaming service, the Instance Manager method of the service plugin may periodically monitor the number of the active HTTP connections of each deployed cache instance to estimate QoE; if it detects that there is a risk of not being able to maintain customer quality constraints, it can invoke the appropriate SO and RO API calls to scale up the service by allocating more compute resources.

Therefore, to offer a new type of service, the following steps are necessary for the system operator:

1. Create the appropriate virtual machine images and register them with the OpenStack image service.
2. Implement and install the service plugin (in our case, plugins are automatically loaded by the SO component).
3. Create a service template and register it with the SIDR component using the SIDR API.

3.3.4 vCDN flavors

A content provider can retrieve a list of the supported flavors by accessing the service catalogue via the CDNaaS customer-facing API. Each flavor is implemented by a service plugin. We note that our prototype already supports various vCDN types, with different operational characteristics (e.g., hosting content origin servers in the vCDN vs. outside it, in other public/private clouds or in the content provider's premises) and resource allocation schemes. These schemes range from the simple unelastic case where a fixed number of resources (specified in the service template) is deployed, only aiming to have presence at all regions requested by the customer and without considering service quality, to a more sophisticated QoE-aware elastic service. In the latter, the compute resource allocation and VNF placement algorithm implemented in the plugin aims to guarantee customer QoE constraints, constantly monitoring service workload and scaling up/down resources to match current end-user demand and minimize operator cost. We should note that the possibilities for further CDN flavors are numerous, and could also even include licensed CDN software, deployed on demand by the operator over the telco cloud/NFV infrastructure with the appropriate amount of resources allocated by the CDNaaS service orchestrator given the expressed end user demand.

3.3.5 Our framework in the context of the ETSI NFV-MANO specification

Our design is in line with the ETSI Network Functions Virtualization-Management and Orchestration (NFV-MANO) [77] spirit and best practices, and the components of our architecture map to functional blocks of the MANO specification: The functionality provided by the SO and the RO is part of the Virtual Network Function Orchestrator (VNFO) component in NFV-MANO, while we use OpenStack as the de facto Virtualized Infrastructure Management

(VIM) suite. IM, in turn, maps to the VNF Manager (VNFM) MANO component, and SIDR implements functionality that would be the responsibility of the VNF Catalogue component.

It should be noted that NFV-MANO allows for various different architectural and functional options [76]. We opted for a split NFVO functionality, where the RO is a separate architectural component which operates as the sole proxy between the VIM and the SO or IM instances, "hiding" the details of the VIM northbound interface. This approach has two advantages in our case. First, by decoupling the NFVO and RO operations, it facilitates future extensions of our scheme towards multi-administrative-domain cases, where the CDNaaS operator can also lease resources on cloud infrastructures managed by other providers (each with their own ROs, potentially managing heterogeneous VIMs), still itself controlling service orchestration. Second, by mediating the communication between SO/IMs and the VIM (*indirect mode*), our design avoids being locked to a specific VIM framework; the underlying technology can change, but all software components except for the RO will be unaffected.

3.3.6 Service deployment time

We present results on the time it takes for a vCDN service instance to be deployed and be operational. Our results are useful for having an estimate of how long it would take for the customer's content (e.g., video steams) to be initially available to its end users, as a function of the size of the vCDN deployment. We also attempt to identify the main factors influencing service instantiation time, which can drive future efforts on fine-tuning and optimizing the performance of our CDNaaS scheme (e.g., by developing algorithms for VM image placement across an operator's data centers).

We set up a cloud testbed with two compute nodes; the cloud controller is collocated with one of the two. We hosted the software components of our CDNaaS architecture on a separate computer, from which the necessary calls to the OpenStack API were performed. To evaluate the effects of the network conditions on service deployment times, we configured the mean round-trip times in the controller-compute and API client-controller paths to d, using the Linux tc/netem utility. A high-level view of our testbed is shown in FIGURE 3.2. In our example setup, following a customer request over our northbound API, a vCDN is deployed with the following service-level components:

- A number (two, in our example) of web proxy/cache VNF instances per compute node (each such node represents a regional Data Center). One load balancer per region is responsible for evenly distributing requests among region-local VNF instances.
- One name server VNF.

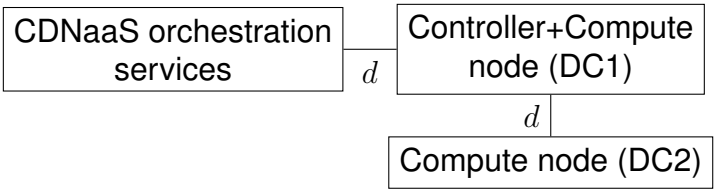


FIGURE 3.2 : Testbed configuration. An OpenStack setup with two compute nodes was created. The OpenStack controller was collocated with one of them. A separate machine was hosting our CDNaaS orchestration services (customer API, SO, SIDR, RO), over which we carried out vCDN service deployment requests. The average RTT in each path was configured to d.

The kvm [53] hypervisor is used to host guest VMs. Each proxy/cache VNF instance is created from a 1.6 GB Debian VM image running nginx, configured at boot time to act as a reverse proxy to an origin server hosted at the customer premises (its IP address and web server port are provided by the customer over the northbound API).

This time to deploy this service is determined by the following factors:

- The time taken to boot and configure the necessary VNF instances, as well as the other service components, such as load balancers.
- The signaling overhead due to the message exchanges between the controller and the compute nodes and between the OpenStack API client (in our case, the RO component) and the controller.
- The time to transfer the VNF images from the image store to compute nodes.

Therefore, significant parameters that influence the service deployment time are the delay in the paths between the entities of our architecture, the size of the VM images to transfer, and the number and amount of traffic of API calls towards the underlying cloud infrastructure that need to be carried out.

OpenStack allows for VM images to be cached locally at compute nodes, which can significantly speed up deployment time. However, there could be cases where an image is not available locally at boot time (e.g., if this functionality is for some reason unavailable, or if it is the first time a specific VM is to be launched on a specific host). Our experiments demonstrate that, in such cases, service deployment time is dominated by the time to transfer the image.

FIGURE 3.3 presents the time it takes for our example vCDN deployment to become fully operational in a single data center. In this case, for each DC, two proxy VNFs are launched

and the load balancer is configured. We compare deployment times for the case where the compute node is collocated with the controller and the image store (local DC) vs. the case for a remote compute node deployment (remote DC), for various link RTT values and when image caching is enabled or not. We notice that when image caching is disabled, the deployment time is dominated by the time to transfer the 1.6 GB image to the remote host. We also observe that even in the case of launching a VM at a host collocated with the image store, deployment times without image caching are noticeably increased. Finally, increased RTT values in the two links also contribute significantly to the perceived delay, since they affect both signalling (e.g., OpenStack API calls) and data (image transfer) traffic. As expected, the effect of the RTT is more important for a remote DC deployment, more than doubling total delay when per link RTT increases from 100 ms to 500 ms.

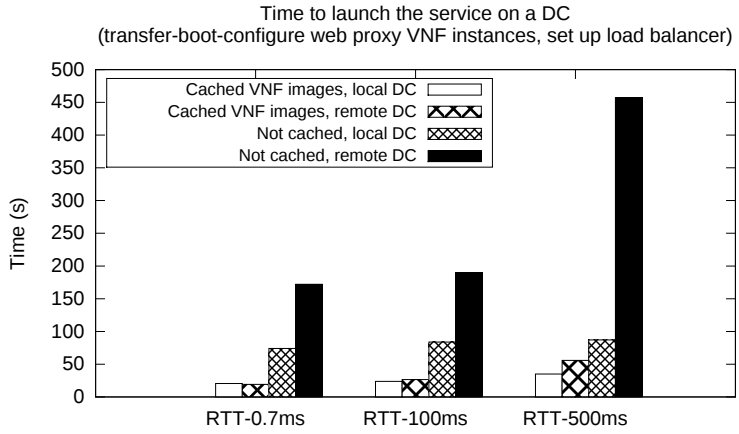


FIGURE 3.3 : Time until a vCDN deployment on a single data center becomes fully operational, under different experimental settings.

In an ideal CDNaaS implementation, the VNF instances per region would be launched in parallel and the SO/RO would monitor their status until they become active. Then, each instance would be registered with the local load balancer. Since OpenStack does not allow such registration requests to be carried out in parallel, we estimate the time it takes for the VNF instances in a specific host/region to become operational as the maximum time it takes for a VM to boot (since they are instantiated in parallel), plus the time it takes for all VMs to be sequentially registered with the load balancer. To evaluate the effect of the number of VNF instances per DC, we first measured the time it takes for a single load balancer member registration request to complete under different RTT conditions, and then estimated the time it takes to deploy the service on a local or a remote DC as the number of such VNF instances increases. Since the time to register a proxy/cache VNF instance with the load balancer ranges from 1.5 s (when RTT = 0.7 ms) to 2.4 s (for RTT = 500 ms), we see a small

but noticeable increase in deployment times, as the number of instances grows. FIGURE 3.4 and 3.5 present these results when image caching is enabled and disabled, respectively.

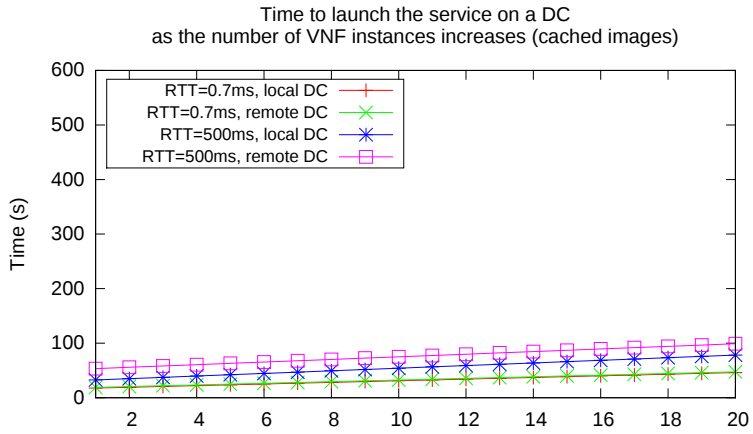


FIGURE 3.4 : Time until a vCDN deployment in a single data center becomes fully operational as a function of the number of VNF instances to be launched, when VM images are already cached.

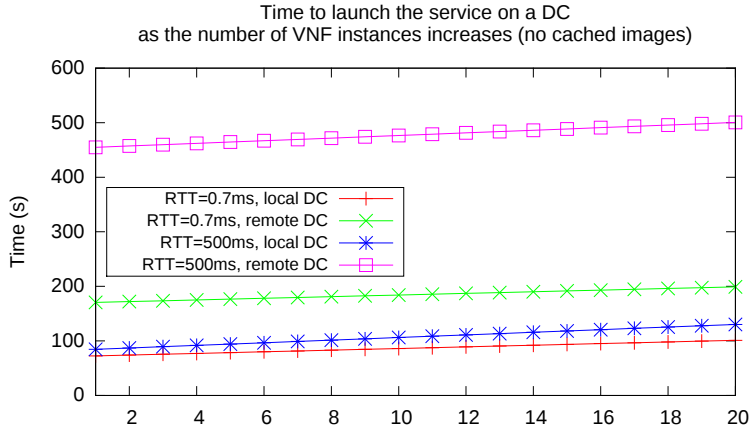


FIGURE 3.5 : Time until a vCDN deployment in a single data center becomes fully operational as a function of the number of VNF instances to be launched, when VM images are not cached. For the case of a remote data center, image transfer times dominate the overall service instantiation time.

3.4 Enabling technologies and their performance

A critical aspect of CDNaaS provision is appropriately balancing between service quality and operational cost. Our proposed design, which offers customers the option to target specific levels of user demand and define end-user QoE thresholds, forces the system to

operate under customer quality and operator capacity constraints. Therefore, resource management (and, in turn, pricing-related) decisions need to be taken with awareness of the capabilities and capacity of the underlying cloud and virtualization technologies.

In the direction of offering the necessary insight to the CDNaaS operator for such decisions, we carry out an extensive testbed measurement campaign, aiming to quantify the performance of key enabling technologies and its relationship with service workload.

Service performance has many factors affecting it: Network capacity and conditions, resources (CPU, memory, storage) allocated to the service, current demand (concurrent number of users accessing the service), the specific virtualization technologies and their configuration, the software used to implement a service, the cloud platform, etc.

In this experimental study, we focus on performance mainly as a function of the available processing resources[3]. Our particular interest is to derive the relationship of virtualized service performance vs. workload on a single virtual CPU (vCPU), our processing unit (or, otherwise put, our unit of scaling), which in our case corresponds to a single CPU core of a physical host.

We present results which can be interpreted under two perspectives: the operator perspective, where the focus is on the performance bounds of the technologies used (e.g., server request throughput), and the user perspective, where our metrics of interest are QoE-oriented (e.g., response time, video viewing experience). For this purpose, we experiment with a generic HTTP service, but also with an HTTP video streaming application.

3.4.1 Virtualization technologies and testbed configuration

We perform a comparative study of two candidate technologies to implement VNFs, each with different characteristics, namely virtualization and containerization. In the first case, we use virtual machines on top of the kvm [53] hypervisor, and, in the second, docker containers [19]. Note that both technologies are supported by OpenStack, our cloud computing software of choice. For a comparison of the features and the internal workings of the two technologies the reader is referred to the work of Dua et al. [23]. On the other hand Felter et al. [28], present a thorough experimental evaluation of their associated overhead for specific applications.

We execute our tests on an HP Z800 workstation with a 16-core Intel Xeon processor and 32 GB of RAM, running Ubuntu 14.10. We are benchmarking the performance of the popular

3. Although equally important, depending on the application, the effects of network resources are not studied here. We will study network-related performance issues in future work. We assume that the operator can tackle them independently of compute resource allocation, e.g., by appropriately provisioning the end user-data center paths.

nginx HTTP server, which is the technology we are using in our proof-of-concept CDNaaS implementation. We have carefully tuned nginx and the operating system (both host and guest) for high performance, and in order to alleviate network and I/O bottlenecks. Nginx works by spawning a number of worker processes, which handle user requests. The optimal strategy is to launch one worker per CPU core available. To deal with large numbers of concurrent users, we increased the maximum number of allowed concurrent connections per worker (and the respective operating system limits on the number of open file descriptors), and set the `tcp_tw_reuse` operating system option, to allow for reusing sockets that are in the TIME_WAIT state.

One of the aspects that we wish to quantify is the scalability of the service as a function of the CPU resources available. Intuitively, this should scale linearly with the number of available virtual CPUs (cores); our intuition is experimentally verified. To achieve the appropriate level of isolation, for each of our experiments, we pin the server VM/container and the load generation tool to separate core sets using the *taskset* utility, and give the highest priority to the respective processes using the *nice* command.

To benchmark our server, we use the *weighttp* tool [112]. Being multi-threaded, it takes advantage of the availability of multiple CPU cores, ensuring that the HTTP traffic generator does not become the performance bottleneck.

Before the experiments, we also verified that network I/O is not the bottleneck. For the experiments using kvm, we activated the *vhost-net* module, which enables in-kernel handling of packets between the guest and the host, thus reducing context switching and packet copying overheads. By using two virtual Ethernet interfaces on the guest and connecting each one of them to a tap device on the host, we measured (using *iperf*) a 30 Gbps aggregate host-to-guest TCP throughput, enough to saturate our HTTP server. For docker, to avoid the overhead associated with Network Address Translation (NAT) in host-to-guest communication, we launched containers using *host*-mode networking. Thus, containers have direct access to the host networking stack.

We also experimentally verified that disk I/O was not a bottleneck, since the file downloaded by the load generator sessions was served from the disk cache. We also repeated our experiments with the web server configured to serve files from a memory-mapped tmpfs file system, and our results were unaffected.

TABLE 3.1 : Startup times for a kvm Debian image and a docker container

	Mean (s)	95% confidence interval
kvm	11.489	(11.178, 11.8)
docker	1.129	(1.124, 1.135)

3.4.2 Startup time comparison

One of the advantages of container technologies is that they are more lightweight compared to VMs. In contrast, to start an HTTP server/cache VNF instance hosted in a VM typically requires booting a full operating system. By startup time we define the time interval from launching a VM/container until it is capable of responding to HTTP. To measure it, we start the VM or the container and simultaneously scan TCP port 80 (where our web server listens to) using *nmap* [81] from the host, until the port is detected open (i.e., the HTTP server is ready). Table 3.1 presents mean startup times for a kvm Debian image and a docker container (average of 50 experiments, 95% confidence intervals). We notice that the startup time for kvm was approximately 11.5 s, while for docker it was an order of magnitude smaller.

It should be noted that in a cloud environment, booting a VNF instance also involves transferring the VM image or container from the image store to the actual host(s) where it will be executed. As we have shown in Section 3.3.6, if image caching is not enabled, this can have a significant effect on the time to launch a vCDN instance.

3.4.3 Performance of a generic HTTP service

We first focus on HTTP performance. Under various experimental configurations, we measure the following quantities: (i) the HTTP/caching server's request throughput, i.e., the number of requests per second that a server can process, and (ii) its response times.

A Request throughput

An important aspect of HTTP server performance is the rate at which they can serve user requests. Being able to estimate the request throughput of a VNF instance with specific processing characteristics allows the CDNaaS provider to calculate the number of instances to deploy to cater for the demand of a specific customer, and appropriately respond to demand dynamics by up/down-scaling the deployment. Request throughput is a function of the processing capabilities of the server, its software architecture, the number of parallel users accessing the server, the size of the files to serve, and its disk and network I/O capacity.

We carried out a set of experiments to measure how our nginx-based HTTP server/cache VNF scales with the CPU processors available and the number of concurrent users. In our first test, to study these two properties in isolation, we minimized the effect of disk and network I/O by having HTTP clients request a special URI which corresponds to a single-pixel transparent GIF (the response body amounts to 43 bytes); this is a resource embedded in the HTTP server and causes zero disk I/O. We then emulated parallel users by varying the number of concurrent HTTP sessions on the client side (weighttp). Each emulated user constantly performed HTTP requests and we measured the HTTP request throughput when 1 or 2 CPU cores were allocated to the server. The results of this test are shown in FIGURE 3.6. We notice that request throughput scales roughly linearly with the number of CPU cores.[4] Interestingly, we notice that docker achieves approximately 10K requests/s more than kvm for large numbers of parallel users. Each point in the figure represents a request throughput value calculated from the execution of 100 million requests.

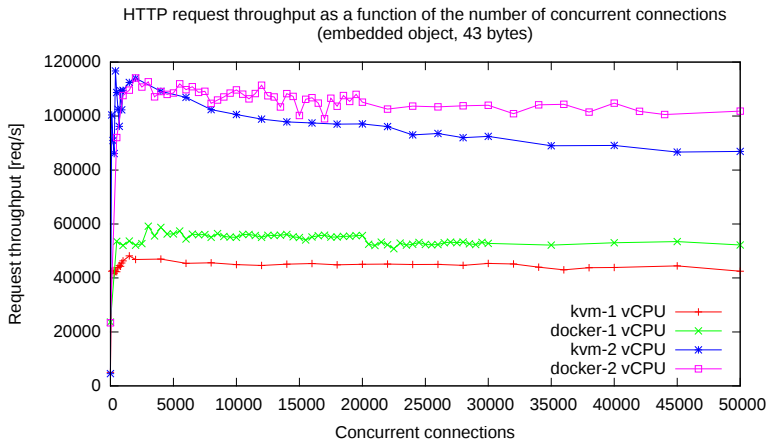


FIGURE 3.6 : HTTP request throughput for increasing numbers of parallel connections. We compare the use of docker and kvm and the impact of utilizing more CPU resources.

We further experimented with HTTP object sizes which correspond to a video service. To select realistic such sizes, we encoded a 720p HD video[5] using H.264/AVC [96], and prepared it for Dynamic Adaptive Streaming over HTTP (DASH) [46] delivery using the DASHEncoder [55] tool. Under DASH technologies, the client receives a Media Presentation Description (MPD) file, which includes information on the available representations (different qualities) of the same video, which is segmented in chunks. Afterwards, the client proceeds

4. Although the figures show only the case for 1 vs. 2 vCPUs, our experiments with higher numbers of vCPUs demonstrate the same linear scaling.

5. We used the Blender Foundation's "Big Buck Bunny" [6] open-source movie. The same video sequence is used in our video QoE tests presented in section 3.4.4.

TABLE 3.2 : Chunk sizes (bytes) for our test H.264/AVC DASH video sequences.

	Min	**Average**	Max
Low quality (250 Kbps)	69852	**73262**	250785
High quality (2500 Kbps)	80755	**434830**	843744

to download the video chunk-by-chunk, potentially switching among the available video qualities (and thus bitrates) to better adapt to network conditions. Table 3.2 shows the average, minimum, and maximum chunk sizes for our test video, for the low (250 Kbps) and high (1.7 Mbps) quality representations. We therefore varied the number of concurrent connections across 100, 1000, and 10000, and measured request throughput when clients were requesting a 73 KB and a 435 KB file.

FIGURE 3.7 presents the achieved HTTP request throughput when all clients are requesting (i) the minimal embedded object (empty GIF, 43 bytes), (ii) 73 KB video chunks (low quality), and (iii) 435 KB video chunks (high quality). We notice the expected response rate decrease as the size of the requested objects increases. One vCPU is capable of sustaining a high-quality video request throughput of ∼8600 requests/s when the HTTP server is containerized, compared to ∼3800 requests/s when it is run in a kvm VM for 10000 parallel HTTP connections. We witnessed a proportional performance increase when the HTTP server was using 2 cores; we have omitted these data from the figure for reasons of clarity.

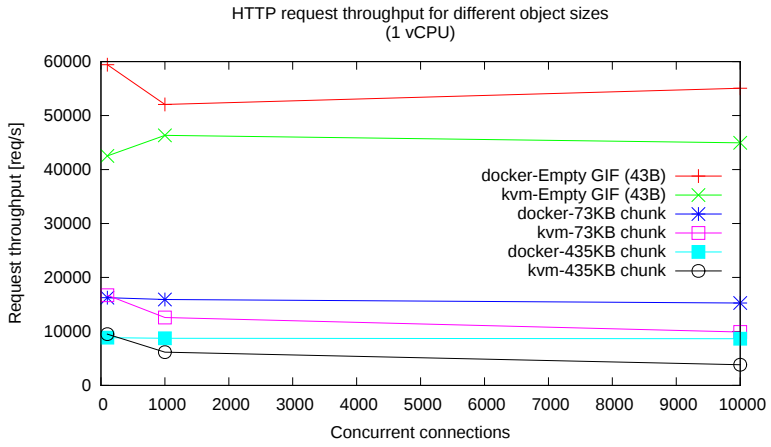


FIGURE 3.7 : HTTP request throughput for various object sizes and for increasing numbers of concurrent connections.

B Response times

Our other performance metric is related with user experience. We measured HTTP response times as a function of the virtualization technology used, the number of parallel

connections, and CPU resource availability. We launched 90% of the concurrent HTTP connections using *weighttp* and we recorded response times for the remaining 10% using *ab* [111], due to its latency measurement capabilities. All figures of this Section present empirical Cumulative Distribution Functions (CDF) of response times; each point represents the percentage of requests which were served in a time less than or equal to a specific value on the x-axis.

FIGURE 3.8 compares a virtualized (kvm) to a containerized (docker) HTTP server, to which 1 processing unit (i.e., 1 core) is allocated. As shown, an increase in the number of parallel connections results in higher latencies, as a result of the connection management overhead on the server. Docker achieves lower response times, since it incurs less overhead for interacting with the operating system and for network I/O (especially given our host-mode configuration, which gives it native access to the host's networking stack). Scaling up processing capacity by adding more vCPUs can significantly reduce latency in both cases.

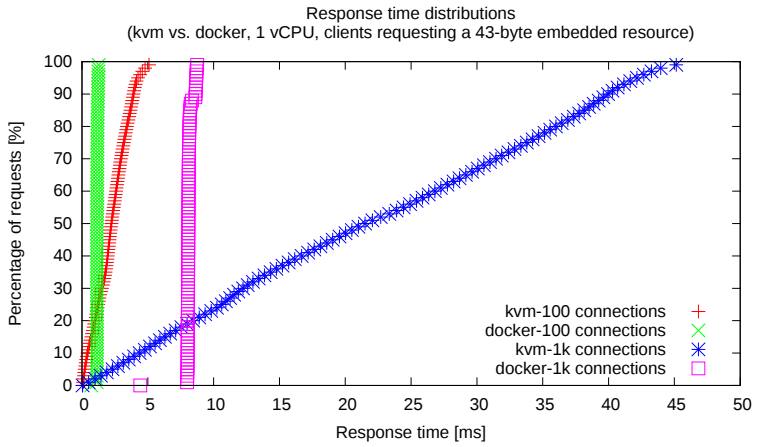
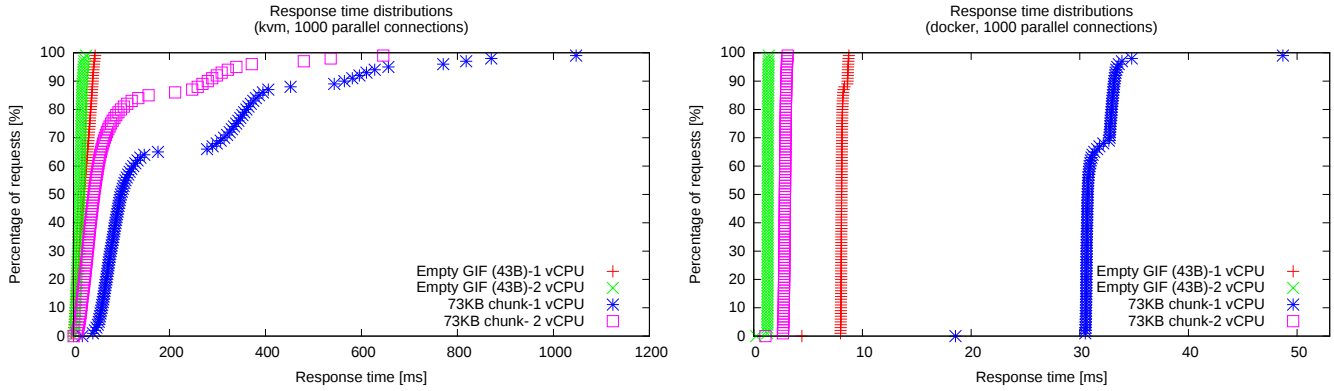


FIGURE 3.8 : Comparison of the response time distributions of a virtualized vs. a containerized HTTP server, for different numbers of concurrent connections. Parallel HTTP connections request a minimal embedded object (43-byte empty GIF file).

When clients request larger objects, response times increase. This increase is noticeable in both cases. For kvm, as FIGURE 3.9a shows, it reaches more than 200 ms for 35% of the requests for a 73K file when the server VM runs on a single CPU core and there are 1000 concurrent connections (star points). Docker (FIGURE 3.9b) performs better: In the same settings, 98% of the requests are completed in less than 35 ms (blue curve, star points). Adding a second vCPU improves performance significantly in both cases (square points).

This performance improvement in terms of latency when we scale up CPU resources is more evident as server load (number of parallel connections) grows, as shown in FI-



(a) kvm (b) docker

FIGURE 3.9 : Response time distributions for different sizes of the requested objects.

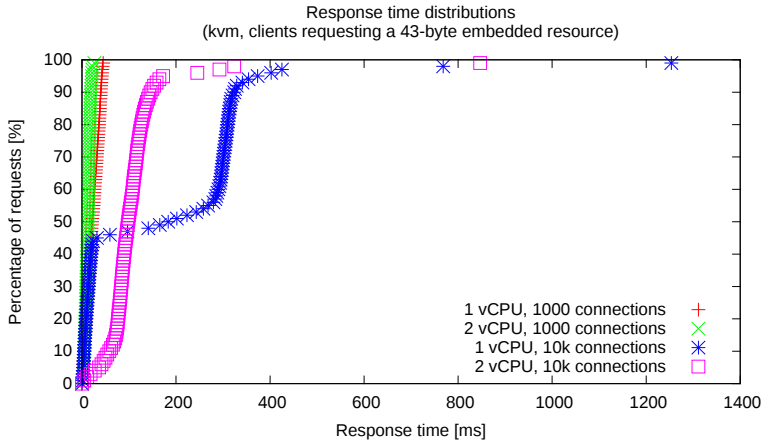


FIGURE 3.10 : Response time distributions for a server hosted in a kvm VM as the number of available vCPUs scales, for different numbers of parallel HTTP connections. Clients request a minimal embedded object (43-byte empty GIF file).

GURE 3.10.

3.4.4 Performance of an HTTP video streaming service

In a similar spirit as in our previous experiments, we measure the performance capabilities of an HTTP video server on a single vCPU in the presence of multiple parallel video sessions. Following a user-centric approach, we empirically quantify the relationship between service workload and QoE in video-service-related terms. We use a vanilla nginx server, which serves user requests for video content prepared for DASH delivery, while on the client side we measure QoE on a DASH video player in the presence of multiple parallel emulated video streams.

A Measurement methodology

Our QoE metric is the Mean Opinion Score (MOS), i.e., the expected rating that a panel of users would give to the quality of the transmitted video in a 1-5 (poor-excellent) scale. To estimate it we apply the Pseudo-Subjective Quality Assessment (PSQA) [93] approach. PSQA consists in training a Random Neural Network (RNN) based on experiments with physical subjects under controlled conditions, where a set of parameters affecting quality is monitored and the ratings of users are recorded. The trained RNN classifier can then be applied in real time and output the expected MOS for specific values of the input parameters. Singh et al. [97] have applied PSQA to estimate QoE for H.264/AVC-encoded video delivered over HTTP, and we are using their tool in our experiments.

The PSQA tool operates on 16-second video segments. As our test video sequences are larger (475 s), we calculate a MOS value for each 16-second window. Since we cannot argue that user experience is independent for consecutive time instances, we maintain a moving average of the calculated MOS in order to capture recency effects according to the following equation:

$$MOS_{i+1} = 0.3MOS_i + 0.7s_{i+1},$$

where MOS_i is the moving average calculated up to window i and s_{i+1} is the MOS sample calculated for the $(i+1)$-th window.

The input parameters for the QoE estimation tool are the following:

- The number of interruptions a $16s$ window.
- The average and the maximum interruption duration.
- The average value of the Quantization Parameter (QP) across all picture macroblocks in the measurement window. QP is an indication of picture quality (the higher the QP, the lower the video bitrate and quality.

These parameters need to be appropriately normalized ; a procedure detailed in [97].

To acquire the required input, we extended the open-source vlc media player with specific monitoring functionality, i.e., with the ability to record QP values, and playout interruptions and their duration. Furthermore, we modified the vlc DASH plugin to disable video rate adaptation, thus only serving the video representation with the selected bitrate.

We further make the following assumptions with respect to QoE calculation:

- If at a specific 16-second window there is an interruption which is longer than 8 s (half of the measurement window), we consider that the MOS is 1.0 (minimum value possible). This is because the RNN used shows saturation when quality is very bad and its training focused on being more accurate when quality is good, at the expense of inaccuracies when interruptions are very large or very frequent [97].

- If the video playout is terminated prematurely or if the video fails to start, e.g., due to connection reset problems (something we observed frequently for large server loads), we consider that MOS = 1.0 for all video windows which follow playout termination.

We use the same DASH test video sequence which we prepared for our previous experiments, with each chunk carrying two seconds of video. In all the experiments that follow, the video client is configured to request only the high-quality video representation (no rate adaptation; mean chunk size of 435 KB; video bitrate of 1.7 Mbps). To emulate multiple simultaneous viewers, we generated parallel HTTP sessions using a modified version of the *wrk* HTTP load generator[6] which allowed for specifying the exact request rate. Each of the parallel *wrk* connections was downloading a 435KB file at the video rate (i.e., 1 chunk/2 s).

At each experiment, which we repeated a number of times to acquire the necessary number of QoE samples for statistical confidence, we varied the number of parallel emulated video sessions, while at the same time recording QoE-related information on a vlc instance accessing a DASH video from the server.

We used the same host/guest configurations as in our previous experiments. Depending on the experiment, we varied the server workload appropriately to find the load points after which interruptions, and thus quality degradation, start to take place. (In other words, we do not report QoE results for all "low" load values, since QoE was unaffected.) We are interested in the performance of the video service when using one vCPU; the results we present when using two vCPUs serve just to demonstrate that the service indeed scales and to verify that performance degradation for high loads is not due to network bottlenecks.

B Results

FIGURE 3.11 presents the average MOS value observed across all $16s$ video samples for each case. We observe that a vanilla nginx server can sustain up to more than 5000 parallel HD video sessions with an excellent quality when kvm is in use and the guest uses one vCPU. For loads of more than 6000 parallel users, video interruptions start to take place, which reduce QoE, especially as load grows. The frequency (FIGURE 3.12) of such playout interruptions follows an increasing trend with server load, reaching, on average, more than 2.5 interruptions per minute when there are 12000 parallel video streams.

6. https://github.com/giltene/wrk2

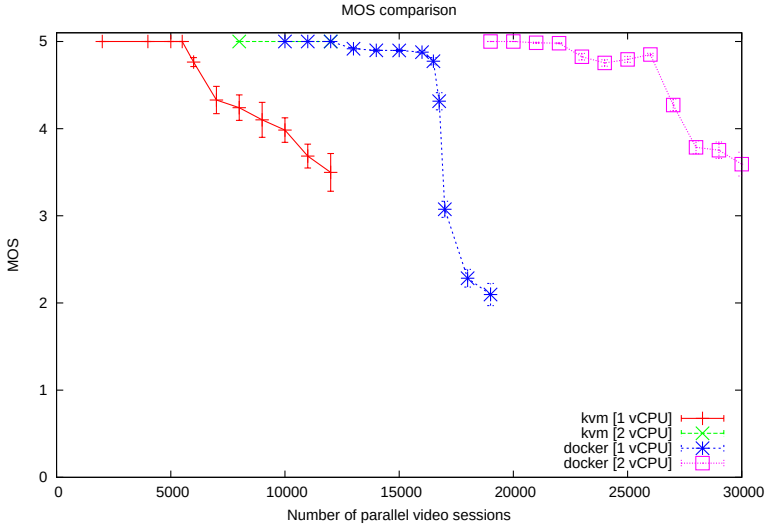


FIGURE 3.11 : Average QoE as a function of the number of parallel users accessing a HD video from a web server. Each point is the mean of a few hundreds of QoE samples (MOS values), presented with 95% confidence intervals.

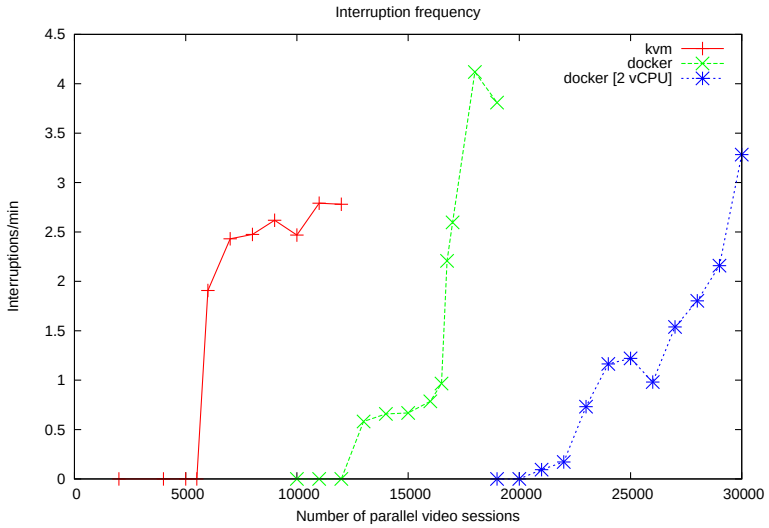


FIGURE 3.12 : Interruption frequency (in interruptions/minute) as server load grows.

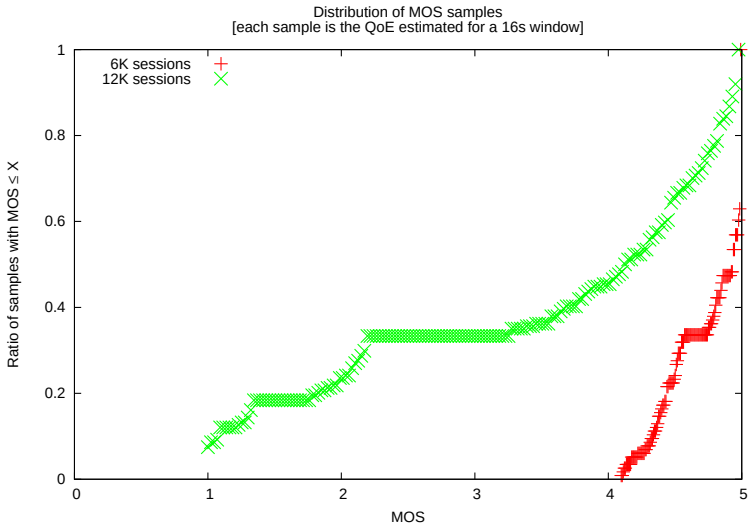


FIGURE 3.13 : Empirical CDF of the QoE of video samples for two different server loads when the service is hosted in a kvm virtual machine.

A similar trend is observed when docker is used. However, the respective figures for docker reveal that for the same compute resources allocated, performance is much better. While for kvm video interruptions start to happen when load exceeds 5500 users, docker on a single vCPU maintains an uninterrupted video service even for double the number of concurrent streams (12000). Performance scales approximately in a linear fashion when adding one vCPU. In this case, a docker container can serve more than 23000 users without interruptions and even when 25000 users are served in parallel from a 2-vCPU container, MOS reaches 4.8 in the 1-5 scale.

We observe that these results are in line with the outcome of our request throughput experiments. For example, in Section 3.4.3 A we found that on a single vCPU docker can sustain 8.6K requests/s for the chunk size that corresponds to the HQ video (see FIGURE 3.7). The fact that, for the same configuration, with for more than 17K parallel video streams (i.e., 8.5K requests/s, given that each chunk carries 2 s of video) MOS starts to more drastically degrade is a user-centric expression of the same phenomenon.

From a user experience perspective, apart from the average QoE observed across all 16-second video samples per case, another interesting metric would be the ratio of viewing time that QoE is above a specific threshold (or, in other words, the ratio of samples in which QoE is above this threshold). This could also be encoded in an SLA of a different format, where the customer is interested in ensuring that end users will enjoy an acceptable QoE level for more than a specific percentage of the video viewing time. To evaluate this metric we resort to the empirical cumulative distribution function of the QoE samples we collected

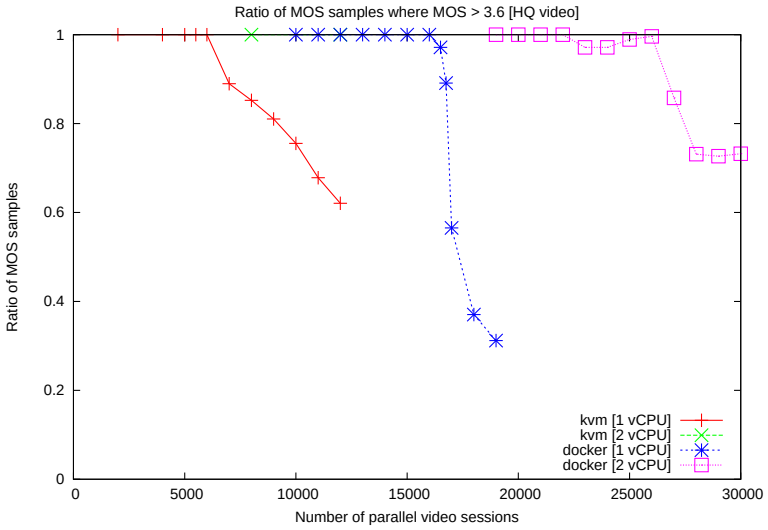


FIGURE 3.14 : Probability that quality in a sample is acceptable (i.e., MOS > 3.6).

in our experiments, examples of which for kvm (1 vCPU) we present in FIGURE 3.13. This interpretation of our results is depicted in FIGURE 3.14, where we present the probability that QoE for a 16-second sample exceeds a MOS threshold of 3.6, which, in various contexts (e.g., VoIP), is considered acceptable quality.

Table 3.3 summarizes our results in terms of (i) MOS and (ii) the probability that QoE for a 16-second sample exceeds a MOS threshold of 3.6. The last two columns of Table 3.3 correspond to (iii) the duration and (iv) the frequency of such playout interruptions, which follow an increasing trend with server load, reaching, on average, more than 7 seconds and 2.5 per minute respectively.

TABLE 3.3 : Summary of results. QoE as a function of the number of parallel users (streams) accessing a video service hosted on a single-core kvm-based web server/cache VNF.

# parallel streams	MOS (average)	Acceptable QoE ratio	Average interruption duration	Interruption frequency
2000	5.00	1.0	0.0	0.0
4000	5.00	1.0	0.0	0.0
5000	5.00	1.0	0.0	0.0
5500	5.00	1.0	0.0	0.0
6000	4.76	1.0	2.99	1.90
7000	4.34	0.89	5.36	2.43
8000	4.24	0.85	6.54	2.47
9000	4.10	0.81	6.61	2.62
10000	3.98	0.76	7.26	2.47
11000	3.69	0.69	7.34	2.79
12000	3.50	0.62	8.55	2.78

3.5 Conclusion

We presented the design and implementation of an architecture allowing a telecom operator to offer a cloud-based vCDN service to content providers on demand. Our approach, which is in line with current standardization efforts on Network Function Virtualization, improves on resource management and service provision flexibility compared to traditional telco CDNs, thus having the potential of empowering the role of telecom operators in the content delivery value chain. Our CDNaaS design is extensible. It supports the deployment of a multitude of different CDN flavors, each with their own features and resource management schemes. For the latter, a better understanding of the performance capabilities of the underlying virtualization technologies as to vCDN provision is necessary. We therefore carried out an extensive experimental evaluation of state-of-the-art containerization and virtualization schemes on top of which a vCDN service can be built, and derived empirically the relationship between workload and service performance under specific technology configurations. With these results as a basis, we propose in the next chapter a method to allocate cloud resources. In particular, we propose uses of our previous results to derive resource allocation/service placement algorithms that aim to optimally satisfy customer demand, minimize on cost and offer availability guarantees.

CHAPTER 4

RESSOURCE MANAGEMENT AND VNF PLACEMENT

4.1 Introduction

Recent studies in networking and cloud computing focus on virtualizing network functions with the aim of providing highly scalable and flexible service deployment, cost reductions, and improved end-user experience. As mentioned in prior chapters ETSI has recently proposed the NFV-MANO architecture [77], which specifies a set of architectural components and interfaces to realize the NFV vision. Among the many use cases envisioned in such an environment is the provision of a vCDN service [79, Use Case #8].

The interactions between CDN providers and network operators [33] have received considerable attention. Many content delivery models involve different degrees of cooperation between stakeholders [43]. The *telco CDN* is one of the studied models, where a telecom operator installs data centers at points of presence in its network and offers a CDN service to content providers. NFV facilitates the deployment of such a service. The operator moves from deploying dedicated hardware for hosting CDN components, such as content origin servers, caches, load balancers, and DNS resolvers, to leasing its telco cloud resources for launching virtual instances (Virtual Machines (VMs) or other containers) of the above components on demand, allocating resources intelligently at the optimal locations in its network, and scaling them to match dynamic workloads.

As presented in the previous chapter, our design allows for the application of a multitude of service-tailored dimensioning, resource allocation, and resource management algorithms via plugins. This chapter focuses on these aspects in particular. A vCDN deployment request generally involves solving the following problems: (i) Deciding on the necessary amount of computing (and other) resources to dedicate in order to efficiently respond to user requests. (ii) Deriving an appropriate placement of virtual CDN service components on the underlying NFVI, with the aim ideally of minimizing cost and offering availability guarantees. Trying to achieve these two objectives can be compromising, due to their conflicting nature; employing

more virtual or physical resources to improve on service availability via redundancy and fault tolerance naturally increases management (e.g., power consumption) cost. It should be noted that the importance of resilience and availability for NFV has been acknowledged by the ETSI NFV Industry Specification Group, which has published specific requirements and guidelines, identifying, among others, the cost-availability tradeoff [78].

We have provided solutions for the first problem by exploiting the experimental result obtained in the end of the previous chapter. The main objective was to combine customer-provided demand dimensioning information (e.g., target number of users/video streams per region) with network and compute infrastructure awareness for optimal resource allocation. For that, we focused on a video content delivery service; we experimentally quantified the relationship between video QoE and service workload, and with this as a starting point, we use this information to (i) first optimally decide on the amount of CPU resources to allocate to satisfy customer QoE requirements while minimizing the cost for the operator. (ii) Second, the distribution of the necessary virtual computing resources to a number of virtual service components and the appropriate placement of the latter in the operator's NFVI. The following contributions are highlighted in this chapter:

- We optimally decide on the amount of resources (vCPU) to deploy and on their configuration, in order to match customer demand.
- We capture the conflicting objectives of service availability and deployment cost by proposing a multi-objective optimization formulation for the problem of joint compute resource allocation and VM placement.
- By relaxing some of the problem's assumptions, we provide polynomial-time heuristic algorithms to solve it and show experimentally that the derived solutions are close to the optimal, while at the same time being significantly faster than an exact algorithm provided by a commercial solver.
- We show our approach to outperform two baseline schemes (random placement and first-fit), as well as a Genetic Algorithm inspired from the state of the art [106] which we have devised as an alternative solution.

This chapter is organized as follows: In Section 4.2 we review the state of the art on VNFs and VMs placement. In Section 4.3 we give the context of which we put our proposed VNF placement algorithms. Based on the experiments, we introduce in Section 4.4 a measurement-driven methodology to determine the amount of CPU resources necessary to satisfy specific QoE levels for a CDN service. The obtained results (number of vCPUs)

are the input of the algorithms presented in Section 4.5 where we propose a model for joint vCPU-to-VM allocation and VM placement. We give heuristic solution for a relaxed version of the latter problem in Section 4.6. Furthermore, we study an alternative way to tackle our problem by proposing an adaptation of a Genetic Algorithm from the state of the art in Section 4.7. We present experimental results on the performance of our scheme compared with heuristics widely used in the literature, the aforementioned Genetic Algorithm, and an exact solution in Section 4.8. We provide notes on the relevance of our work with ETSI NFV and on network resource allocation in Sections 4.9 and 4.10 respectively. We conclude the chapter in Section 4.11.

4.2 State of the art

The placement of virtual instances to physical hosts is central in NFV in particular, and cloud computing in general. The main objective is to find a suitable set of physical machines (PMs) with enough capacity to host the virtual instances with specific resources allocated to each one of them.

In NFV, these instances host VNFs, and are implemented as VMs or other container technologies. A suitable VNF placement is mostly motivated by maximizing resource utilization or overall performance, while minimizing cost in terms of energy consumption, network traffic or penalties associated with Service Level Agreement (SLA) violations under various constraints [20, 91, 37, 44].

In [88], Fabio Lopez Pires and Benjamin Baran classified the most relevant VM placement algorithms in the literature by proposing an up-to-date survey and presented a VM placement taxonomy helping to determine research opportunities.

We complete the optimization approaches of VM placement problems reported in this survey in the following.

- *Mono objective*: functions that consider only one objective function (or individual optimization of more than one objective function, one at time) 64% of [88] studied papers proposed this method to solve the VM placement. An example is the work of Ibn-Khedher et al [45], which propose a mono-objective formulation of migration cost in order to minimize the VMs migration cost during the placement.

- *Multi-objective*: a well known optimization approach where the problems are formulated as mono-objective approaches, 32% of the studied articles propose this approach. The way of formulating as mono-objective approach simplified the VM placement problem.

- *Pure multi-objective*: rarely used approach, it includes a defined number of objective function and decision variable as well as constraints, the method is based on the

concept of Pareto dominance [1], we use the latter in our optimization formulation of the problem (detailed in Section 4.5).

Referring to this paper [88], we can have an idea on how to tackle the problem of allocating resources, virtual or physical. This work shows the importance of VM placement problems in cloud computing, they also show how the problem can be taken from different angles and criteria, depending on the scenarios. The paper helped us to identify the taxonomy of VM placement and adapt it to our VNF placement.

Mann [66] reviews different VM placement models and algorithms, citing, among different formulations, the problem of packing the VMs into a minimal number of PMs, considering PM capacities and the load of VMs. This reduces to the bin packing problem which is NP-hard. Various heuristics to solve it in this context exist, such as first-fit, where the items (VMs) are placed in the first suitable bins (PMs) [8, 98].

Li and Qian [58] survey different network function orchestration frameworks. In particular the authors compare different network function placement strategies and highlight the positive and negative points of different approaches of VNF placement.

The common ground in the way various flavors of the VM placement problem are tackled is the need to provide heuristics to problems with typically high computational complexity. Minimizing the number of physical hosts to place VMs for cost and performance optimizations, a procedure known as VM consolidation, may have an adverse effect on service resilience and availability in the face of PM failures. Contrary to the aforementioned works, service availability is our special focus.

In our study of the state of the art regarding VNF/VM placement , various algorithms to solve the VM placement problem are proposed.

One well known algorithm is the Ant Colony Optimization algorithm, it uses positive feedback mechanism called "ant colony". This is because it imitates the behaviour of ant colonies looking for food, the algorithm is inspired by the way it communicates with each other by pheromone left on past paths. This algorithm inspires Yongqiang Gao and al [36] to solve their multi-objective VM placement problem. Their algorithm aims to efficiently obtain a set of non-dominated solutions that simultaneously minimize total resource wastage and power consumption. In particular, their problem is formulate as a multi-objective combinatorial

1. In multi objective optimization the concept of dominance helps to identify when a solution is better than the others. The optimal solution is known as a Pareto front which is a set of solutions in a single/mono objective optimization.

optimization problem, aiming to, at the same time, optimize total resource wastage and power consumption. The authors assume that if two VMs are running on the same server, the CPU utilization of the server is estimated as the sum of the CPU utilization of the two VMs. The resource wastage model is based on making an effective use of the resources in all dimensions and balance the resources left on each server along different dimension. This is defined as the potential cost of wasted resources (using the ration of used resource to total resource). The authors also study and propose a model for the power consumption which takes into account the state of the server, i.e, if the latter is fully used or idly used (when the server is off). Their model consider some constraints similar to our model, such as the assignment of one VM to only one server, and the capacity of the server (memory, CPU) [21].

Jain et al. [48] propose an approach using virtualization technology (kvm) to allocate data center resources dynamically based on application demands supporting green computing [2] by optimizing the number of server actively used. Their resource allocation problem is defined as the bin packing problem where each server correspond to a bin and each VM is an item to be packed. Globally the system architecture is composed by virtual machine monitor such as kvm [53] and multiple VMs share the same PM, and application such as DNS, web server, etc. run inside a VMs. The VM Management (VMM) handles resource according to a PM such as CPU utilization, memory, etc. which are delivered to a centralized VM scheduler cited as the most important component in the architecture. It receives as inputs the resource requirement needed to launch a VM, the capacity and the load of PMs, and the current mapping VMs-PMs. The VM scheduler has modules to estimate the resource demand in near future and to optimize VMs layout so that the resource demand are satisfied while minimizing the waste of resource.The approach of [48] use an exponentially weighted average called EWMA to predict and observe the load. According to the authors, the prediction part is important in improving the stability and performance of the resource allocation decisions. The prediction is out of the scope of this thesis.

Jenn-Wei and Chien-Hung Chen [62] studied the interference-aware VM placement problem which not only fully exploit the resources of PMs but also consider the quality of service requirements of user application and the VM interference reduction. The system model used in their work consists of single controller node and a number of physical machine nodes (similar to the architecture of Xen cloud platform [3]). The controller node principally handle the

2. Green computing is the environmentally responsible and eco-friendly use of computers and their resources. Green computing also study how computing devices can reduce their environmental impact".

3. www-archive.xenproject.org/products/cloudxen.html

resources of PM and the VM placement in a given PM. The integration of an interference-aware in a VM placement has as objectif to not violate the QoS requirements of the applications running in VMs. Their algorithm takes into account (i) the resource demand of VM. (ii) The application QoS and (iii) the interference among VM, the three conditions are in the cloud provider side. The problem is formulated as an Integer Linear Problem (ILP) one. Since the ILP is an NO-complete problem (greedy in computational time), the approach proposed by [62] is to use a heuristic algorithm with polynomial time to solve the VM placement with interference aware. The ILP problem is formulate as an objective function to maximize the profit of the cloud provider after placing the new VMs in PMs by monitoring the service time of network I/O request to verify the QoS constraint. The heuristic algorithm is based on a scenario composed principally of the available capacities of PMs, the resource demands (CPU, memory, storage), rental prices of VMs, and the profits relative to all possible placement relation between VMs and PMs. The operational part of the algorithm are (i) contention phase which is the part of calculating the profit of all VMs, the PM select the VM with the best profit, and a tool is used if one VM is selected by more than one PM by choosing the better profit, (ii) the placement phase is the part where the VM is placed in a chosen PM. (iii) In the reformation part, an update is made to retrieve the VM placement status.

As in our performance comparison where we compare our algorithms with intuitive ones, the authors in [62] compare their proposition in terms of profit, number of QoS-violation VMs and penalty payment, with the random-fit, first-fit and least fit algorithms. The result of the interference-aware VM placement show small number of VMs that violates the QoS compared with the intuitive algorithms. The penalty payment is also minimized, their results show that it is possible to maximize the cloud provider's profit and reduce VM interference by placing a larger number of VMs.

Others such Dieye et al. [18] focus on guaranteeing a minimum service delay to the end-user in the placement towards CDN. The objective is to place the VNFs such that the optimal number of the virtual instances reduces the cost and not violate the delay guarantee.

Regarding the traffic, albeit important, is outside the scope of this thesis. However, we review few works that cover the networking aspects in VM placement. Meng et al. [68] propose a two-tier approximation algorithm to solve the traffic problem in VM placement, the authors prove the NP-hardness [57] of the problem by reducing the problem to the Balanced Minimum K-cut Problem [94] known as NP-hard. They propose an algorithm that minimizes the traffic through the switches, and place each VM in a specific PM by taking as inputs the cost of PMs communications and the network topology of the data center. Others such as [29] propose a network-aware placement of virtual components (computing and data)

of a multi-tier applications in data center. This problem is formulated as combinatorial optimization problem and the objective is to minimize the network overhead of the data center network infrastructure.

In the recent literature on Service Function Chaining (SFC) [58] problems, it has been noticed that the structure of those problems are similar to the problems of VM placement. The authors compare different NFV frameworks and summarize how the design space affects the placement. We agree with the fact that all actual VNF placement are at off-line servers.

On the other hand Li and Qian [59] considered that the traffic steering technique used to enforce the SFC may cause forwarding loops.To this end the authors propose a VNFs placement on the path of each traffic flow, instead of modifying the flow path in order to enforce the policy chain. Their VNF placement problem is formulated mathematically and they propose an annealing algorithm to solve it.

Mijumbi et al. [69] propose an evaluation of resource allocation algorithms considering parameters such as successful service mappings, total service processing times and revenue or cost under various network conditions. In particular, they define cost as the total amount of physical resources that are used by a given mapping and scheduling. As defined by the authors, the difference between revenue and cost is that the revenue only consists of the processing time of the functions, while the cost involves time gaps that are left unused due to functions waiting for their assignment.

On the other hand, Bari et al. [4] propose an optimization problem formulated as an ILP named the VNF orchestration problem (VNF-OP), which consists in minimizing the cost (deployment cost, energy cost, and cost of forwarding traffic), penalties for Service Level Objective (SLO) [4] violations, and resource fragmentation. The constraints taken into account are related with server/link capacities and service chaining. In our work we consider cost as the fixed managerial overhead of operating a virtual machine (resp. physical machine), which is not a function of its workload.

As confirmed by the previous references, the ILP has been a popular modelling technique for VM allocation problems. However, experimental results from Rankothge et al. [92] show that using ILP to find an optimal configuration can have a very long execution time even for a small number of network functions. The authors thus studied approximations by means of

4. SLO is considered as a key element in SLA, it involves specific measurement performances of the Service Provider.

finding the best fitted solution according to a Genetic Algorithm (GA) model of the problem. In particular, they proposed a Genetic Programming-based approach to solve the VM allocation and network management problem, exploring a fixed amount of generations. Although their GA may not provide the optimal solution, it can compute configurations orders of magnitude faster than ILP. GAs are a part of evolutionary computing and were introduced as a computational analogy of adaptive systems [71, p. 64–75]. In the same line Chamarro et al. [12] propose a Genetic Algorithm for the resolution of a brocker-oriented VM placement problem in a dynamic environment. The objective is to maximize on infrastructure capacity while minimizing in migration overhead cost.

The relevant state of the art also includes the GA of Xu and Fortes [106], which aims to solve VM placement formulated as a multi-objective optimization problem of simultaneously minimizing the total resource wastage, power consumption and maximum thermal dissipation. With this work as a starting point, in Section 4.7 we propose a GA tailored to our problem settings. The heuristic algorithms we present in this chapter, however, are shown to outperform our GA both in terms of approximating the optimal solution and execution time.

Yang et al. [109] focus on reliable VM placement, and, in a similar spirit to our case, model service availability as the probability that a subset of the requested VMs is operational. However, they address a different VM placement problem, which, among others, does not consider the distribution of the resources (in our case vCPUs, in their case storage) to be committed, assuming fixed VM resource specifications, and does not take into account VM-level failures.

Ecarot et al. [26] present a method for cloud resource allocation which aims to satisfy both consumers' (or end-users') and providers' interests. Their model is an ILP which minimizes the costs and service unavailability, the latter corresponding to the penalties for quality of service degradations or violations by the providers of a service, which are in turn associated with excessive service workload. This is contrary to our approach, where availability is defined as the probability that a virtualized service is accessible. To deal with the complexity of the problem, the authors apply evolutionary algorithms.

Other researchers, such as Casazza et al. [11] propose to guarantee the availability of a service by replicating VNFs across multiple servers. They apply a probabilistic metric for availability but, contrary to our work, they aim at maximizing the minimum service availability considering the latter as their sole objective under server capacity and other constraints.

Qu et al. [90] focus on the problem of VNF chaining and aim to minimize the network-wide communication bandwidth for the operation of VNF chains under service reliability constraints. Their model does not consider failures at the VM level in its reliability functions and their focus is rather on chain-specific issues such as routing and VNF ordering, which constrain VNF-to-host placement. Furthermore, their model does not include CPU capacity constraints.

Regarding cost modelling, Callau-Zori et al. [10] experimentally quantify how the number of VMs deployed impacts both energy cost and performance. This work verifies our model assumption that cost is linear to the number of VMs and PMs utilized.

Ouarnoughi et al. [84], on the other hand, focus on storage cost. They propose a detailed cost model for IaaS infrastructures which takes into account at the same time various parameters, such as VM execution and storage migration, system storage performance, power and wear out. They also consider cloud SLA violations and the associated penalties. Our cost model is more abstract, encompassing all related sub-costs in linear functions of the number of PMs and VMs utilized. Also, it is not considering the operational cost due to service workload, since this cost is assumed proportional to the workload and in any case independent on the number of PMs or VMs deployed, thus not influencing the selection of an appropriate virtual function placement.

In the context of vCDNs, Ibn-Khedher et al [45] propose an optimal VM placement and migration algorithm. They consider various constraints such as system load, network infrastructure inputing the network topology, and user satisfaction to optimally place and migrate the VMs among the servers. They propose a mono objective model to minimize VM migration cost. Albeit it relevance, we did not consider the VM migration in this thesis.

A relevant aspect is fault recovery to maintain high cloud service availability. To tackle this issue Israel and Raz [47] propose approximation algorithms with performance guarantees and heuristics. This topic is outside the scope of our work.

A classification of existing papers dealing with the VNF placement problem proposed in the state of the art is presented in Table 4.1, according to the different optimization criteria used in each model.

As can be seen in Table 4.1, most of the works in the relevant literature focus only on a single objective, knowing in advance the number of VNF instances to place and having as

TABLE 4.1 : State of the art classification according to the criteria considered.

Network operation/management and other costs	[26], [92], [4], [44], [45], [29], [12], [18]
Energy consumption	[20], [91], [10], [106]
Availability	[90], [26], [11] [18]
Failure recovery	[47]
SLA violation	[37], [84], [44]

an objective to place these virtual instances on physical hosts. Rare are those works which treat the subjects of availability and cost jointly, and there lies the distinctive characteristic of our approach.

4.3 Cost- and availability-aware VNFs placement context

Although our architecture is fairly generic and could support heterogeneous types of services,[5] we focus on the specifics of a video content delivery service. Shifting our attention from infrastructure- to service-level features, we present some specific characteristics and assumptions regarding the operation of the vCDN application.

The CDNaaS operator has data centers in a number of regions, on which vCDN instances for video distribution can be deployed. The dimensioning and placement algorithms are to be executed per region included in the customer request. This is because the target of the operator is to deliver content to each region's end users from the local data center. We assume that vCDN traffic is handled exclusively by the vCDN instance components assigned to a specific region and thus resources are allocated in a region-local manner. Issues regarding redirecting user traffic across regional data centers are outside the scope of this thesis. For each region, a sufficient number of virtual CPUs needs to be allocated for cache (streaming) servers to cope with the expected service demand and QoE constraints. The vCPUs are distributed to a number of VMs which are identical from a functional perspective for reasons of fault tolerance, but also because it may not be possible to consolidate them in a single VM, due to potential capacity limitations of the underlying physical host. From a performance viewpoint, we assume that the number of VM replicas used to deliver a service does not have any effect; only the number of vCPUs allocated to handle the service workload matters.[6]

5. For example, apart from supporting various different CDN flavors, we have successfully used our scheme to orchestrate the deployment of a virtualized Evolved Packet Core network, in a 4G mobile network context.

6. In practice, using a load balancer to distribute user requests for content can have a performance effect (e.g., increased latency) compared to serving requests directly, without the intervention of the load balancer. However, we consider these performance effects negligible in our study.

In this chapter, we pay a particular attention to the fault tolerance of the vCDN service. By design, and not taking into account other service components such as load balancers and DNS servers, the video service is considered available in a region if at least one VM hosting a cache is accessible to users. This is due to the load balancers which transparently redirect user requests to available local cache instances. It should be noted that service availability defined as such does not guarantee that the required QoE levels are met in the event of failures, since the latter depend on the number of compute resources available for responding to user demand. If a VM fails and appropriate repair activities are not in place, the CPU resources allocated to this VM become unavailable. However, with appropriate service monitoring and management schemes, this issue could be adequately addressed: The released CPU resources could be automatically and transparently reallocated to a running VM on the same host, if such one already exists (scaling up), VMs on other physical hosts covering the same region could be scaled up if there is available CPU capacity in their hosts, or, as a last resort, the failing VM could be relaunched on the same host. Such self-healing/repairing techniques are outside the scope of this article.

In this work we assume that the resource allocation procedure is composed of two main phases. First a resource dimensioning algorithm which decides on the optimal allocation of vCPU resources per region needs to be executed, aiming to satisfy the constraints dictated by the customer request and the operator's capacity for a given service demand. For a video streaming vCDN service, these can be expressed in terms of a maximum number of concurrent users that will be accessing the video service per region, respecting the minimum QoE constraint set by the customer, which takes the form of a minimum MOS (defined in section 3.4.4 A) for specific video characteristics (e.g., high definition video).

The second phase uses as input the number of vCPUs calculated in the first phase and derives an appropriate assignment of them over a number of VMs to be deployed on a subset of the available telco cloud physical hosts.

4.4 A QoE-aware virtual CPU resource allocation algorithm

The experiments presented in chapter 3 provide insight on the capabilities of the enabling technologies for our CDNaaS vision. This can be applied to devise appropriate resource allocation and management strategies. Our findings indicate that virtualization comes with a larger overhead compared to using containers. However, it should be noted that, despite their performance and flexibility benefits, containerization technologies for cloud computing have only recently gained popularity, while virtualization can be considered more mature in this context. This helped us to identify a uses of such quantitative results which we present in this section. We show how our experimental results on video QoE vs. service load can be

used to derive a computing resource allocation for a vCDN instance.

4.4.1 System model

We assume that the service provider (SP) has presence in N regions and a content provider (CP) wishes to deploy a virtual CDN over a subset of them. The CP (customer) request includes a demand $D = \{d_1, d_2, ...d_N\}$, where d_i is the maximum number of users simultaneously accessing the service[7] in region i, as well as the respective target quality specification $Q = \{q_1, q_2, ...q_N\}$, where q_i is the desired QoE[8] for end users in region i. The SP needs to decide on the appropriate number of CPU to allocate per region to satisfy the demand with the requested QoE. Our resource unit is a virtual CPU (vCPU) which in practice corresponds to a single CPU core.

We formulate the problem as an optimization one. The objective is to find an assignment $X = \{x_1, x_2, ...x_N\}$ which minimizes the sum of the number of vCPUs x_i allocated to each region i following a customer request, taking into account (i) that the quality delivered to the end user must be better or equal to the quality that the customer requested for, and (ii) that the number of vCPUs deployed must not exceed the per-region capacity $C = \{c_1, c_2, ...c_N\}$, where c_i is the number of vCPUs available at the data center of region i.

Therefore, our problem is formulated as follows:

$$\underset{X}{\text{minimize}} \quad \sum_{i \in N} x_i \tag{4.1}$$

$$\text{subject to} \quad x_i \leq c_i \qquad \forall i \in N \tag{4.2}$$

$$U(x_i, d_i) \geq q_i \qquad \forall i \in N, \tag{4.3}$$

where $U(x_i, d_i)$ is the expected QoE for the end users in region i when the demand is d_i and x_i vCPUs are allocated to cover it. We further make the assumption that if the requested resources for a single region exceed its capacity, then the problem has no solution. Table 4.3 summarizes our notation.

7. To express demand we use the terms “users” and “video streams” interchangeably, implying one video stream per user terminal.

8. Depending on the scenario, this could be expressed in terms of the expected MOS or the percentage of viewing time when QoE is higher than a specific quality threshold. Our algorithm is equally applicable to both cases.

TABLE 4.2 : Summary of notation for QoE-aware vCPU resource allocation

N	Number of regions
$X = \{x_1, x_2, ...x_N\}$	Number of vCPUs allocated per region i
$Q = \{q_1, q_2, ...q_N\}$	Requested quality specification per region i
$D = \{d_1, d_2, ...d_N\}$	Demand specification (maximum number of parallel streams) per region i
$C = \{c_1, c_2, ...c_N\}$	Resource capacity vector (available vCPUs per region i)
T	Load-Quality matrix

4.4.2 An algorithm for the minimum cost solution

In this section, we present an algorithm to solve (4.1), i.e, to find the minimum sum of resources to serve a customer request under capacity (4.2) and quality (4.3) constraints. Apart from the demand, quality specifications and constraints, the input to our algorithm is T, a 2-dimensional matrix with table format 3.3. The first column (load) contains the number of parallel video streams handled by a single vCPU, and the second (quality) contains the QoE that can be achieved under this load. We derive this table from measurements (as shown in Section 3.4.4).

Our algorithm operates as follows: For each region, the minimum number of resources necessary so that the quality constraint is not violated is calculated. For this, the entry j in T which corresponds to the maximum number of parallel streams that can be handled without violating this constraint is located (line 4). Then, the algorithm calculates the number of vCPUs for region i by dividing the demand value (customer demand) by the load value of entry j (line 5) and rounding the result up (we do not allow fractional resources or load sharing across regions). The intuition is that since we aim at minimizing cost, we wish to pack as many streams as possible on a single vCPU; Line 5 guarantees that $U(x_i, d_i) \geq q_i$. Line 6 will ensure that the capacity constraint is not violated.

The number of vCPUs obtained is the minimum that can be deployed under the specific problem settings and assumptions. If in a single region we reduce by 1 the number of vCPUs, the demand cannot be handled and the quality constraint is violated, while if we add 1 vCPU,

the derived solution will be suboptimal, given that our algorithm aims only at satisfying quality constraints, and not at optimizing for quality.

Algorithm 1 Minimum total number of vCPUs

INPUT : N, Q, D, T
OUTPUT : $X = \{x_1, x_2, ...x_N\}$ ▷ Assignment which minimizes $\sum_{i=1}^{N} x_i$
for each region i **do**
    $j = \arg\max_k \{T_{k,1} \mid T_{k,2} \geq q_i\}$ ▷ Row in T with the maximum load not violating the quality constraint
    $x_i = \lceil d_i / T_{j,1} \rceil$;
    if $x_i > c_i$ **then** ▷ Capacity constraint
        return NULL ▷ No solution exists;
    end if
end for
return X

4.4.3 Performance evaluation

To demonstrate the importance of QoE awareness in resource dimensioning decisions, we compare our proposed mechanism for vCPU allocation with a QoE-unaware baseline strategy. This strategy allocates a fixed number of resources for a given service demand, i.e., allocates one vCPU per F streams. For example, if the customer demand for region i is $D_i = 4000$ simultaneous streams and $F = 1000$, the baseline will allocate 4 vCPUs. Since this mechanism does not take into account the amount of resources necessary to satisfy the customer demand with a specific quality level, it can lead to under- or over-provisioning. In other words, depending of the value chosen for F by the operator, it may either lead to the allocation of fewer resources than necessary to attain the customer's desired quality level, or to an excessive amount of resources, which will lead to unnecessary operational costs.

Our algorithm, on the other hand, optimally addresses this quality-cost tradeoff, being aware of the relationship between load and user experience. To demonstrate this advantage, we simulate both strategies on a real PoP topology. We use the publicly available POP map of the Greek Research and Technology Network (GRNET) [38], which has presence in 49 cities, and assume that the operator has deployed data centers with a specific vCPU capacity at each one of them, which can host CDNaaS VNF instances. In our simulation, the capacity of each region ranges from 100 to 900 vCPUs. We vary the value of F for the baseline mechanism and calculate the total number of allocated vCPUs for a specific customer demand (number of parallel video streams/users), which is selected, for each region, uniformly at random from the $(500, 20000)$ range. Note that the baseline scheme does not

take into account the customer quality constraint, which is fixed to 4.5 for all regions. Then, for the same customer request (same demand/quality specification), we run our algorithm.

FIGURE 4.1 shows the results of our simulations. The x-axis represents the total number of vCPUs allocated (cost), and the y-axis represents the respective MOS (quality). The square point corresponds to the outcome of our algorithm (1210 vCPUs in total, resulting in an average MOS value of 4.76 across all regions), while the cross points represent the performance of the baseline scheme, starting from $F = 12000$ and down to $F = 1000$ users/vCPU. (Note that the value of F, i.e., the number of video streams served per vCPU is not shown in the figure.) The flat line is the quality constraint. The figure shows that our algorithm outperforms the baseline scheme: The latter achieves a comparable level of quality only for $F < 6000$, when it decides to allocate at least 1612 vCPUs. Conversely, the baseline scheme satisfies the quality constraint only for $F < 7000$, in which case it allocates 1397 vCPUs.

The baseline scheme performance for decreasing numbers of users per vCPU reflects the tradeoff between quality and cost: The more the resources to allocate to handle a specific load, the better the quality. Our QoE-aware algorithm addresses this tradeoff optimally, offering both user-centric (improve QoE) and operator-centric (save on resources) benefits.

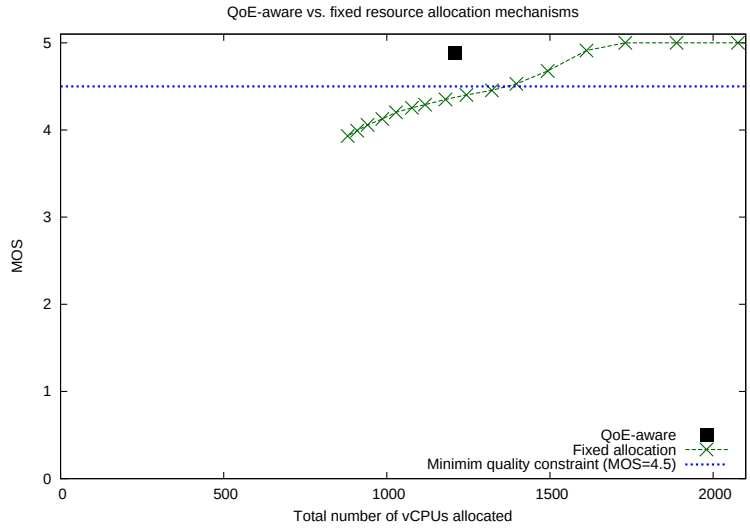


FIGURE 4.1 : Comparison of a QoE-aware with a fixed resource allocation mechanism. Depending on its configuration, the baseline scheme may lead to under- or over-provisioning. Our QoE-driven approach, on the other hand, derives an assignment which minimizes cost respecting the quality constraint.

Our algorithm also runs efficiently. For the GRNET topology, including 318 POPs (regions), it takes less than 10 ms to calculate the minimum vCPU assignment when executed on a

single CPU core of an Intel Xeon E5-1603 processor. The running time of our algorithm scales linearly with the number of regions, as shown in FIGURE 4.2.

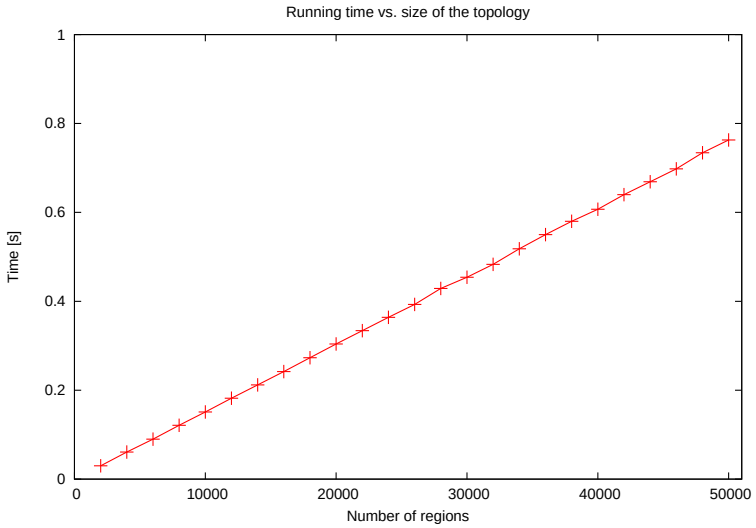


FIGURE 4.2 : Execution time of our QoE-aware algorithm for increasing topology sizes (i.e., numbers of regions/POPs.) Each point is the mean value of 50 iterations.

4.5 A model for joint vCPU-to-VM allocation and VM placement

As mentioned previously, the resource allocation procedure is composed of two main phases. The first phase presented in the previous section where we propose a mechanism for the CDNaaS provider to optimize the amount of compute resources to allocate to guarantee the desired video QoE levels. In particular, this mechanism calculates the minimum number of vCPUs needed to serve a given customer-specified demand under capacity (operator) and quality (customer) constraints.

The second phase, which is the main focus of this section, uses as input the number of vCPUs calculated in the first phase (previous section) and derives an appropriate assignment of them over a number of VMs to be deployed on a subset of the available telco cloud physical hosts. How exactly these resources are allocated and how the respective virtual instances are placed is a matter of addressing the tradeoff between availability and cost: Splitting resources across instances and distributing the latter across multiple physical hosts can result in higher service availability, but also increased operational cost (e.g., power consumption) for the operator, which may be reflected in the service price. In this work we respond to the

latter by proposing a multi-objective optimization formulation for the joint CPU resource allocation and virtual instance placement problem, where customer preferences are translated to specific weightings for the two conflicting objectives. Note that the two resource allocation phases are independent. The model and algorithms presented in this article could be directly applied in conjunction with any algorithm for deriving the number of vCPUs necessary for a service request.

4.5.1 Preliminaries

We aim to assign p vCPUs, the output of the first phase of the resource allocation procedure, to a number of up to p virtual machines (VMs), and the placement of the latter in (a subset of) the available m physical machines (PMs) [9] of a regional data center, aiming to minimize the deployment cost and to maximize service availability, while respecting PM vCPU capacity constraints. The outcome of this process is a matrix $X = (x_{ij})$ with $i \in [1, p]$ and $j \in [1, m]$, where x_{ij} denotes the number of vCPUs assigned to VM i hosted in PM j. The upper bound for i is the number of vCPUs to assign (since our unit of processing is a vCPU, we cannot have more VMs than the number of vCPUs to assign). The calculation of the optimal assignment should respect physical capacity, cost, and availability constraints.

4.5.2 Cost model

We consider that the deployment of a VM comes with a fixed management overhead which is not a function of its workload. For example, this cost can account for the energy consumed for booting the VM or for operating other system- or service-level components (e.g., operating system). We further assume that for each PM which hosts service instances, there is a fixed overhead which is not a function of the PM workload nor the number of VMs hosted by it (e.g., energy cost for keeping the physical machine in an operating state, overhead of various system-level components). We model the above costs as linear functions of the number of VMs and PMs utilized by a service deployment, which is in line with the experimental observations of Callau-Zori et al. [10].

In matrix X, the number of non-zero elements represents the number of VMs deployed. Therefore, the cost of an assignment X at the VM level is given by

$$C_V(X) = e_V \sum_{i=1}^{p} \sum_{j=1}^{m} \mathbb{1}(x_{ij} > 0), \tag{4.4}$$

9. In the rest of the thesis, unless otherwise noted, the terms *host*, *PM*, and (physical) *server* are used interchangeably.

where e_V is the fixed cost per deployed VM. In a similar spirit, the cost of an assignment X at the PM level is determined by the number of PMs that host at least one VM. This corresponds to the number of columns in X that contain at least one non-zero element. This cost is thus given by

$$C_P(X) = e_P \sum_{j=1}^{m} \mathbb{1}(\sum_{i=1}^{p} x_{ij} > 0), \tag{4.5}$$

where e_P is the fixed cost per used PM, and the overall cost of an assignment follows:

$$C(X) = C_V(X) + C_P(X). \tag{4.6}$$

4.5.3 Availability model

We define service availability as the ability of the system to offer at least a minimal service, i.e., to have, at any time, at least one VM accessible, which implies that at least one PM should be up to host the respective VM(s).

We make the following assumptions:

- A VM i can fail with probability $q_i^{(V)}$, independently of the other VMs and PMs, and irrespectively of the load imposed on the VM.
- Each PM j can fail with probability $q_j^{(P)}$, independently of the other PMs or the load imposed on it. The above probabilities are assumed to be known to the operator as a result of measurement studies, prior experience, or other historical information. (The same applies to VM failure probabilities.)
- If a PM fails, all VMs deployed on top of it are assumed to fail because of that.

Therefore, a VM may become inaccessible either because it fails or because the PM that hosts it fails. VM failures can be correlated due to their dependence on the underlying PMs. Based on this, we define a *correlated group* of VMs as the VMs which are executed on the same PM. For a correlated group to be available, the following conditions should hold:

- The PM is up, and
- At least one of the VMs deployed on the PM does not fail.

The probability that a correlated group deployed on PM j is available is thus given by:

$$a_j = (1 - q_j^{(P)})(1 - \prod_{i \in [1,p] | x_{ij} > 0} q_i^{(V)}) \tag{4.7}$$

For a vCDN service deployment to be available, at least one correlated group should be available. Since correlated groups fail independently, the probability that a vCDN service deployment is available is given by

$$\begin{aligned} A(X) &= 1 - Pr\{\text{All correlated groups fail}\} \\ &= 1 - \prod_{j\in[1,m]|\Sigma_{i=0}^{p} x_{ij}>0} (1-a_j) \\ &= 1 - \prod_{j\in[1,m]|\Sigma_{i=0}^{p} x_{ij}>0} \left[q_j^{(P)} + (1-q_j^{(P)}) \prod_{i\in[1,p]|x_{ij}>0} q_i^{(V)} \right] \end{aligned} \tag{4.8}$$

Since, by construction, any feasible solution includes at least one PM with at least one VM assigned to it, both product terms in (4.8) are over non-empty sets.

4.5.4 Problem formulation

The aim of the system operator is to derive an optimal assignment X^* which minimizes cost while maximizing availability. These two criteria are conflicting: the more the VMs deployed and the PMs used to host them, the less the risk of service unavailability, but, at the same time, the higher the cost of the deployment. Since it is not possible to optimize for both criteria simultaneously, we apply a scalarization approach to transform the problem to a single-objective one. The relative importance of the two criteria in deriving an optimal assignment is dictated by a specific *policy*, which is encoded in a pair of weights w_a and w_c (resp. availability and cost) such that $w_a + w_c = 1$. Given a specific policy, the system operator derives the optimal solution to the following problem:

$$\underset{X}{\text{minimize}} \quad w_c C(X) - w_a A(X) \tag{4.9}$$

$$\text{subject to} \quad C(X) \leq E \tag{4.10}$$

$$A(X) \geq A \tag{4.11}$$

$$\sum_{j=1}^{m} \mathbb{1}(x_{ij} > 0) \leq 1, \forall i \in [1,p] \tag{4.12}$$

$$\sum_{i=1}^{p} \sum_{j=1}^{m} x_{ij} = p \tag{4.13}$$

$$\sum_{i=1}^{p} x_{ij} \leq C_j, \forall j \in [1,m]\,. \tag{4.14}$$

To deal with the potential difference in the magnitude of the two components of the objective function, the values of $C(X)$ and $A(X)$ are appropriately normalized in the $(0,1)$ interval using the upper-lower-bound approach [67]. This model supports specific maximum

cost and minimum availability constraints (C and A, respectively; see (4.10) and (4.11)). Constraint (4.12) ensures that, for any VM, its vCPU resources are allocated on a single PM, since a VM cannot be split. Note that an assignment can result in this sum being zero for multiple values of i. This occurs when the number of VMs to deploy is less than the number of vCPUs to assign, a case typical in practice. Constraint (4.13) ensures that all vCPUs are allocated, and constraint (4.14) guarantees that the CPU capacity of each PM is not exceeded.

Resource allocation and virtual function placement problems such as the one that we address in this work are typically shown to be hard to solve by reduction from known packing or knapsack problems [66]. Our case has the particular properties that (i) the function to minimize involves multiple objectives, (ii) it is non-linear and non-separable due to the availability component that it includes and its weighted combination with the cost objective, and (iii) involves a non-linear constraint (4.11). Such characteristics are known to make such problems harder to tackle [40]. For example, the non-separable integer knapsack problem with the additional restriction that the objective function has quadratic form is already NP-hard [35, 7]. Moreover, we should note that our availability model is generic in the sense that it allows for heterogeneous PMs with different failure probabilities. Even for the simpler version that we consider in the rest of this article, where PMs are assumed identical from a reliability perspective, thus having equal failure probabilities, we show experimentally that deriving exact optimal solutions is computationally expensive by solving the problem using the CPLEX optimizer.

Table 4.3 summarizes our notation used in this section.

4.6 Solving a relaxed version of the problem

Finding an exact optimal solution to the problem can be prohibitive computationally. We thus propose an efficient heuristic algorithm to derive solutions which we show to be near-optimal in Section 4.8. In particular, instead of jointly deciding on optimally distributing the number of vCPUs to VMs and placing the latter in PMs, we tackle the two subproblems separately in two stages: First, we decide on the appropriate number of VMs to utilize and, second, on their actual placement on PMs and the CPU resource distribution to them. Both steps of our *Two-step Resource Allocation and Placement (TRAP)* heuristic algorithm can be efficiently solved in polynomial time.

Therefore, the overall procedure of deriving an appropriate resource allocation and placement following a customer's CDNaaS instance deployment request involves three sub-

TABLE 4.3 : Summary of notation for joint vCPU-to-VM allocation and VM placement

p	number of vCPUs to allocate
m	number of available PMs
v	number of VMs
$C(X)$	cost function
$A(X)$	availability function
C_j	capacity of a physical machine j
w_a	availability weight
w_c	cost weight
e_V	fixed cost per VM instantiated
e_P	fixed cost per PM utilized
$q_i^{(V)}$	failure probability of VM i
$q_j^{(P)}$	failure probability of PM j

problems outlined below:

1. Find an optimal number of vCPUs to allocate per region to serve customer demand. We solve this problem using the procedure presented in the previous section.

2. Decide on the optimal number of VMs to launch under a specific policy, satisfying cost and availability constraints.

3. Place those VMs on an optimal number of physical hosts and distribute vCPUs among them using a suitable placement algorithm, having as input the results of the previous steps (optimal number of VMs, number of vCPUs). Specific host capacity, cost, and availability constraints apply.

Sections 4.6.1 and 4.6.2 detail our methods to solve the second and third subproblems respectively. We address each subproblem independently, providing for each one a problem formulation and an algorithm to optimally solve it.

We should note that, for each subproblem, the same minimum availability constraint value (A) as in the original problem is applied, although the availability functions themselves may differ. For the management cost, there is a specific budget for each subproblem (E_V and E_P for the VM- and the PM-level problems respectively), such that $E_V + E_P = E$. The system operator is free to decide how the overall budget is split and various methods for this may

be possible. For example, a split ratio $s \in (0, 1)$, such that $E_V = sE$ and $E_P = (1 - s)E$, can be selected by defining a parameter $\epsilon \in (0, 1)$ and successively solving the two subproblems for each $s \in [\epsilon, 2\epsilon, \ldots, 1)$, eventually selecting the assignment which minimizes (4.9). This procedure involves $\lfloor 1/\epsilon \rfloor$ executions of the proposed algorithms for the two subproblems. If this method is applied, the operator can tune ϵ to force a specific number of iterations.

4.6.1 Deciding on the number of VMs to launch

In this section, we describe our solution to the second subproblem, which consists in finding the optimal number of VMs to launch. We define an objective function to minimize, which includes a management cost and an availability component.

A Availability

Given a known VM failure probability q_V, assumed to be fixed and identical for all VMs of the same type included in a vCDN instance (VMs of a specific type, e.g. streaming servers, are considered identical and their failure probability does not depend on the resources allocated to them nor their workload), we define service availability at the VM level as

$$A_V(x) = 1 - q_V^x, \tag{4.15}$$

where x is the number of VMs to deploy. This expression for availability corresponds to the probability that at least one VM is available and is heuristic in the sense that it ignores PM failures and considers VM failures independent.

B Management cost

The management cost at the VM level is given by (4.4) and is assumed linear on the number of VMs used. A simplified expression defined in terms of the number x of VMs to deploy is

$$C_V(x) = e_V x. \tag{4.16}$$

C Problem formulation

Applying the specified policy expressed as the combination of weights (w_c, w_a), the function to minimize at the VM level becomes

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & w_c C_V(x) - w_a A_V(x) && (4.17)\\ \text{subject to} \quad & A_V(x) \geq A, && (4.18)\\ & C_V(x) \leq E_V, && (4.19)\\ & x \geq m_{min}, && (4.20)\\ & x \leq p, && (4.21) \end{aligned}$$

where A, E_V are the respective constraints of availability and cost, and m_{min} the minimum number of PMs capable of accommodating the required number of vCPUs. The latter is given by

$$m_{min} = \min\{l \in [1, m] | \sum_{i=1}^{l} C_i \geq p\}, \tag{4.22}$$

where $C_1 \geq C_2 \geq C_3 \geq .... \geq C_m$ are the capacities of the m PMs of a given region in non-increasing order.

The objective (4.17) is to find an optimal number of x under availability (4.18) and cost (4.19) constraints.

Constraint (4.20) ensures that x is such that no single VM is assigned more vCPUs than the maximum single-host available capacity. If x is not large enough, and given that the number of vCPUs to distribute to these VMs is fixed, and at most x PMs will eventually be used (we cannot use more PMs that the VMs to deploy), it could happen that there exists no subset of x PMs with enough aggregate vCPU capacity (i.e., at least p). This would mean that at least one VM would need more vCPUs than even the highest-capacity PM could offer. For instance, when x=1 (with p vCPUs to distribute), if there is no physical host with a capacity superior to p, such a solution is infeasible. On the other hand, (4.21) ensures that the VMs to be launched will not be more than the number of vCPUs to allocate.

D Selection of the optimal number of VMs

In order to find the optimal number of VMs by solving (4.17) under a specific policy, we propose an algorithm whose inputs are (i) the total number, p, of vCPUs to be allocated across the region's hosts in order to serve the customer request, derived in the first phase, (ii) the constraints per objective (A, E_V), and (iii) the combination of weights (w_c,w_a) that defines the adopted policy.

We first observe that the objective function is convex and the value that minimizes its continuous version with $x \in \mathbb{R}_{>0}$ is $x_0 = \log_{q_V} \frac{w_c e_V}{w_a \ln q_V^{-1}}$. Constraints (4.18) and (4.19) can be rewritten as $x \geq \log_{q_V}(1 - A)$ and $x \leq \frac{E_V}{e_V}$, respectively. Constraints (4.18) and (4.20) thus provide a lower bound to the optimal value of x. Which of the two constraints is more restrictive depends on the configuration of the problem. Similarly, an upper bound is provided by the cost constraint (4.19), and (4.21).

To minimize (4.17) we calculate the value $x_0 \in \mathbb{R}_{>0}$ which minimizes the continuous version of (4.17). If x_0 lies within the feasible region defined by the above bounds, we select the closest (feasible) integer value to x_0 as the optimal. Otherwise, we select the value of the constraint that x_0 violates.

We should note that in other problem settings where the objective function is not convex, an optimal solution can be found in pseudo-polynomial time in p by evaluating the objective function and the constraints for each $x \in [m_{min}, p]$. The case $x = m_{min}$ corresponds to a deployment which minimizes the number of deployed VMs, while $x = p$ to a deployment that performs a one-to-one vCPU-to-VM assignment. This algorithm runs efficiently for realistic values of p.

4.6.2 VM placement and vCPU distribution

With the number of VMs v to launch in place, we need to decide on their suitable placement on the underlying PMs and the distribution of the vCPUs among them. The output of this step is an assignment $X = (x_{ij})$, with $i \in [1, v]$ and $j \in [1, m]$, which represents a heuristic solution to (4.9).

Similar to how we treated the sub-problem of selecting an optimal number of VMs, we define appropriate availability and cost functions at the PM level as follows.

A Availability

To measure a solution's availability, we use (4.8), assuming identical PMs with a known failure probability fixed to $q_P^{(j)} = q_P, \forall j \in [1, m]$.

B Management cost

Assuming identical PMs, and in turn a fixed management overhead for each PM on which we are placing the customer's VMs, irrespective of their number and the service workload

imposed, we model PM-level cost as a linear function of the number of PMs eventually used for hosting at least one VM. This cost is given by (4.5), substituting p for v in the upper limit of the second summation, since the number of VMs in this case is already known, as calculated at the second step of our scheme.

C Problem formulation

We propose the following multi-objective formulation for the problem of placing v VMs to a subset of the available m PMs and distributing p vCPUs to them under capacity, availability and cost constraints, and given policy (w_c, w_a):

$$\underset{X}{\text{minimize}} \quad w_c C_P(X) - w_a A(X) \tag{4.23}$$

$$\text{subject to ÌĄ} \quad A(X) \geq A, \tag{4.24}$$

$$C_P(X) \leq E_P, \tag{4.25}$$

$$\sum_{j=1}^{m} \mathbb{1}(x_{ij} > 0) = 1, \forall i \in [1, v] \tag{4.26}$$

$$\sum_{i=1}^{v} x_{ij} \leq C_j, \forall j \in [1, m] \tag{4.27}$$

where A, E_P are the respective availability and cost constraints.

Constraint (4.26) ensures that a VM is allocated to only one PM, and the set of constraints (4.27) guarantees that the available capacity per PM is not exceeded.

D An algorithm for VM placement and vCPU distribution

We propose the following algorithm to derive an optimal solution to (4.23) in polynomial time, whose input is (i) the number, p, of vCPUs to be allocated, (ii) the policy and constraints per objective, (iii) the sorted PMs in non-increasing order of capacities, i.e, $C_1 \geq C_2 \geq C_3 \geq .... \geq C_m$, (iv) m_{min}, and (v) the number, v, of VMs; the latter three are obtained in the previous step.

This algorithm consists in creating and evaluating a candidate assignment of VMs to m' PMs, for each $m' \in [m_{min}, \min(m, v)]$. More than v PMs are not necessary, since that would directly mean that at least one PM would not host any VM. We observe that the value of the objective function only depends on the number of PMs used and the number of VMs assigned to each PM. Our algorithm starts by carrying out the following steps for each m':

1. Distribute the v VMs among the m' first PMs proportionally to their capacities.
2. Evaluate the objective function and the constraints for the derived assignment.

Our algorithm returns the assignment X of VMs to PMs which minimizes the objective function for the given policy, or responds that there is no feasible solution under the given constraints.

This procedure has $\mathcal{O}(m \times \min(m, v))$ complexity: The two steps have to be repeated at most $\min(m, v)$ times. Step 1 takes time linear on the number of PMs. (Using the *largest remainder* apportionment method for proportional distribution is $\mathcal{O}(m)$.) Evaluating the availability function could also be implemented to take $\mathcal{O}(m)$ time, if we separately keep track of the number of VMs assigned per PM in a candidate solution.

If a feasible solution using m^* PMs is found, the algorithm ends by:

1. Distributing the p vCPUs among the first m^* PMs proportionally to their capacities.
2. For each of the first m^* PMs, distributing the number of the allocated vCPUs evenly to the VMs assigned to it.

The complexity of the algorithm is dominated by the first two steps. Finally, since only the number of PMs and VMs, and the placement of the latter influence the objective function value, any method of sharing vCPUs to PMs/VMs is equivalent.

4.7 A Genetic Algorithm as an alternative approach

A question that may naturally emerge is whether applying *meta-heuristics*, such as Genetic Algorithms, could offer high-quality solutions with reasonable computational cost. A GA generally operates as follows. Each potential solution to the problem is encoded as a string (chromosome) of properties (genes) and is characterized by a fitness value, which is an expression of the solution's quality. Starting from an initial population of candidate solutions (chromosomes), a GA iteratively applies genetic operations on them to produce new generations of better quality. Genetic operations include crossover, where new chromosomes are generated by selected parents, and mutation, where a single chromosome is randomly altered.

In this direction, we build on the approach of Xu and Fortes [106] who, as in our case, deal with a related multi-objective VM placement problem. In particular, they propose a Grouping Genetic Algorithm (GGA) to overcome the limitations of standard GAs when dealing with grouping problems and solve the problem of instantiating VMs on a set of physical hosts

aiming to simultaneously minimize resource wasteage, power consumption, and maximum thermal dissipation. These objectives are conflicting, since consolidating VMs in a small number of PMs may lead thermal dissipation to increase significantly in loaded PMs, albeit minimizing the first two objectives.

Inspired by the GGA of Xu and Fortes and adapting it to our problem settings, we have appropriately modified it to integrate our cost and availability objectives. Furthermore, Xu and Fortes apply a fuzzy logic system as a fitness function to jointly consider the conflicting objectives. In our case, however, the preferences with respect to the objectives are known a priori and are expressed using the combination of weights (policy). We thus apply directly our weighted objective function of (4.9) as the fitness function. Moreover, their algorithm takes as input the number of VMs to place. To obtain this input, we execute the first phase of our heuristic which determines the number of VMs to deploy and then use the GA to select their placement on PMs. After having decided on the number of VMs (see Section 4.6.1), our adapted version of the GA starts by creating an initial population of chromosomes by generating S random assignments of VMs to PMs. For each solution in S, each group of VMs of a PM represents a gene. Then, the algorithm operates iteratively for G generations.

In each generation, as Xu and Fortes suggest, the *ranking-crossover* operator is applied, which is an improved version of the classical crossover. In particular, the advantage of ranking-crossover lies in the fact that it does not select genes blindly from the selected parents, but instead attempts to pick the ones which are ranked higher according to the average value of specific *efficiency* functions defined per objective, thus expected to produce higher quality offspring.

In our context, for minimizing cost, a good solution uses a minimum of virtual and physical resources, while maximizing availability requires instantiating more VMs and spreading them across more PMs. For each ranking-crossover operation, the algorithm selects two random chromosomes as parents and calculates the efficiency values for each involved gene (i.e., group of VMs assigned to a PM). In our algorithm, the efficiency function is defined as the euclidean distance between the pair of values of availability and cost for a gene, given respectively by (4.15) and (4.16) with x representing the number of VMs contained by the gene, and a *centroid*. We define the centroid as a "utopian" point which represents the pair of cost-availability values considered ideal (and probably not attainable by a single solution at the same time; e.g., a solution with minimum cost and maximum availability). The efficiency function (*object-centroid distance*) of a gene is thus given by

$$\mathcal{E}(c, a) = \sqrt{(u_c - c)^2 + (u_a - a)^2}$$

where the centroid (u_c, u_a) is the vector of ideal values of the cost (u_c) and availability (u_a) objectives and (c, a) represents the obtained cost and availability values for the gene.

Our algorithm ranks the genes of the parents in increasing order of the object-centroid distance and creates a new chromosome by combining the highest-ranking genes, i.e. the ones with the smallest values of $\mathcal{E}$.

For each generation round, this procedure is repeated according to the desired crossover rate, r_c, which determines the number of new chromosomes to be generated per round. Typically, the crossover rate is set to a value between 0.5 and 1, producing $r_c S$ offspring on average. The eventual solution pool population at the end of a generation is constructed by evaluating the fitness function for each chromosome and keeping the top-S of them. The algorithm is terminated after G generations, where the solution with the minimum value of (5.6) is returned. According to the observations of Xu and Fortes, mutation does not offer tangible advantages in their case. We therefore chose not to perform mutation either.

Xu and Fortes report that the complexity of their Grouping Genetic Algorithm is $\mathcal{O}(SNlogN+NSG)$, where N is the number of VMs to place. The first term is due to the execution of the First-Fit placement algorithm to generate the initial pool of S solutions and the second for generating and evaluating all potential placements for G generations.

Our GGA has a slightly different complexity. Instead of performing First-Fit, we generate the initial pool of S solutions by randomly placing N VMs over the m PMs in $\mathcal{O}(SN)$ time. Then, each of the G generation rounds involves $r_c S$ ranking-crossover operations and the selection of the population of the next generation. For each of the two parents involved in a crossover, calculating the efficiency of up to m genes (in case all m PMs are utilized) requires $\mathcal{O}(m)$ time, sorting the genes of the parents according to their efficiency is $\mathcal{O}(mlogm)$, and creating a new offspring is $\mathcal{O}(m)$. This is repeated $r_c S$ times yielding an $\mathcal{O}(Smlogm)$ time per generation. Finally, selecting the top-S solutions of the population which also includes the offspring requires evaluating the fitness function, i.e. computing the objective function of (5.6), for each solution in the population. This be carried out in $\mathcal{O}(m)$ time (see Section D) per chromosome, while selecting an (unordered) set of the top-S chromosomes of the population to create the next generation is $\mathcal{O}(S(1+r_c)) = \mathcal{O}(S)$. The computation time for each generation is thus dominated by the creation of offspring and the overall complexity of our GA is $\mathcal{O}(SN + GSmlogm)$.

4.8 Performance evaluation

We evaluate the performance of our algorithms under two perspectives. First, we aim to demonstrate how our model and algorithms address the cost-availability tradeoff and show how the selection of specific policies drives the solutions towards different optima. Second, we compare the performance of our scheme (TRAP) with alternative approaches in terms of the quality of the produced solutions (i.e., how close they are to the optimal) and the exe-

cution time. Note that for the latter we implemented our model in the CPLEX[10] environment and used its optimizer to derive exact optimal solutions as a benchmark.

4.8.1 Cost vs. availability tradeoff

We begin with an experiment where we run our algorithms under different combinations of weights (w_c,w_a), each corresponding to a different policy and for the same problem settings. In particular, we use (i) the same number of vCPUs to allocate, (ii) the same number of available hosts and their capacities, and (ii) the same cost and availability constraints. We set the availability constraint to "five nines" (99,999%), a popular availability target for carrier-grade NFV. We fix the overall cost constraint to $E = 160$ and assume that the costs for a single VM and PM are $e_V = 1$ and $e_P = 1$ respectively. The failure probabilities for VMs and PMs are set to $q_V = q_P = 0.001$. Unless otherwise noted, these values are used all across the experiments of this section. We consider a system with 50 PMs, each with a capacity between 2 and 15 vCPUs selected uniformly at random.

FIGURE 4.3 presents the solution space, where the x-axis and y-axis represent the absolute values of the cost and availability functions respectively. Each filled square point corresponds to the outcome of the TRAP algorithm (a vCPU-VM-PM assignment identified as the optimal) according to a specific policy, while empty squares are feasible but suboptimal solutions. The vertical and horizontal lines are the respective cost and availability constraints and limit the space of feasible solutions. Note that in our tests it is possible that for two different policies the same optimal solution is derived. These points are superimposed in the figure.

The selected solutions represent what our algorithm identifies as the *Pareto frontier*, which, in our context, is the set of solutions for which it is not possible to improve on cost without sacrificing on availability and vice versa. Each such solution represents a different availability-cost trade-off. A cost-centric policy, e.g. $(w_c, w_a) = (0.8, 0.2)$, guides our algorithm to select as the optimal a low-cost solution, which, according to the cost function, uses a small number of VMs/PMs. In turn, this corresponds to a lower service availability. (The algorithm guarantees that it is a feasible solution, not violating the 0.99999 availability constraint.) Availability is increasingly higher (≈ 1) when more hosts are used. Such solutions correspond to increasing w_a values, which our mechanism takes into account to drive the calculation towards an appropriate VNF placement which uses more virtual instances and PMs.

10. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

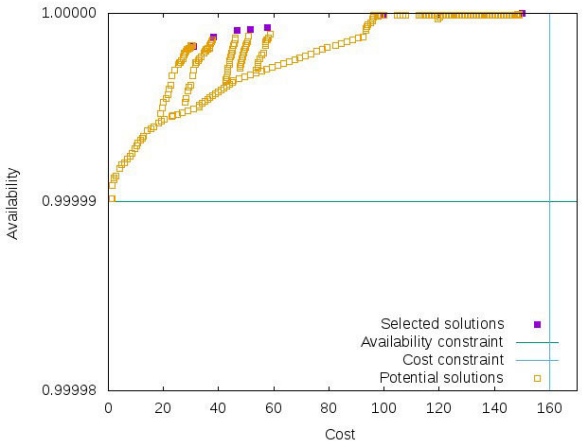


FIGURE 4.3 : Solution space. The straight lines represent cost and availability constraints. The filled boxes are the assignments selected as optimal by our TRAP algorithm under different policies, while the empty boxes are feasible but suboptimal solutions.

This is more evident in FIGURE 4.4, where we detail the evolution of the value of each objective (cost and availability) in each optimal solution as a function of the given policy. For reasons of clarity, the values presented are normalized by mapping the lowest and highest value per objective to 0 and 1 respectively. The figure indicates that as the weight of the cost objective w_c increases, both functions are decreasing. This is positive from a cost but not from an availability perspective, and is due to less VMs/PMs being used as w_c approaches 1 and w_a approaches 0. (The inverse holds for availability.)

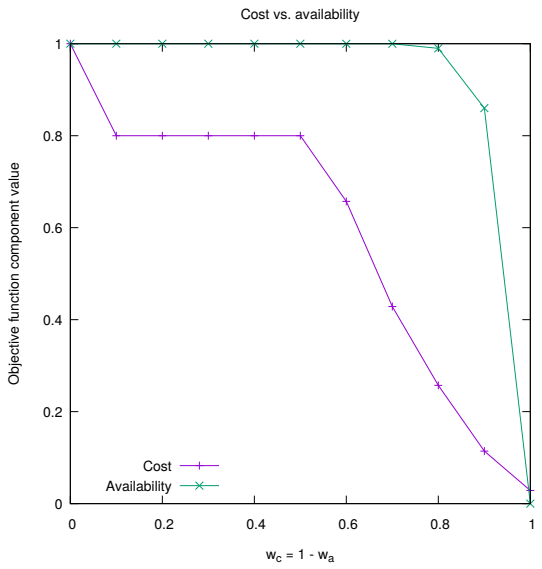


FIGURE 4.4 : Availability and cost as a function of the selected policy.

4.8.2 Performance comparison

We compare TRAP with the following placement algorithms:

- *Random:* Given a number of VMs, *Random* places each VM at an available host selected uniformly at random.
- *First-Fit (FF):* This algorithm places each VM at the first available machine that has the capacity to host it.
- *Genetic Algorithm (GA):* This is the grouping Genetic Algorithm we proposed in Section 4.7.
- *CPLEX:* This is an exact algorithm which derives the optimal solution to (5.6) using the CPLEX optimizer.

Random, *FF* and *GA* require the number of VMs to place as input. In the experiments presented in this section, it is implied that this input is provided by executing the first step of TRAP. This improves the performance of these schemes compared to a random or policy-unaware decision for the number of VMs to place. As our results show, though, our two-step heuristic still brings about significant performance gains.

In our comparison, our metric for the quality of a solution is the ratio of the objective function value of the solution returned by the algorithm in question to the respective value of the (exact) optimal solution returned by CPLEX. This is expressed as a percentage and is denoted as *optimality*. *CPLEX* corresponds to an optimality of 100%.

FIGURE 4.5 presents a comparison of TRAP (purple curve, cross points) with our grouping Genetic Algorithm (green curve, “x” points), random (blue curve, “*” points), and first-fit placement (yellow curve, square points) in terms of optimality as a function of the number of available PMs.

To obtain these results we ran the five schemes under the same configuration, which corresponds to the same policy (in this case, $w_c = w_a = 0.5$), the same number of PMs, each with available capacity selected uniformly at random and ranging from 2 to 15 vCPUs, $p = 700$ vCPUs to distribute, and the same capacity constraint. Each point is the mean of 50 iterations for the same configuration except for PM capacities which vary, presented with 95% confidence intervals.

As can be seen from FIGURE 4.5, TRAP approximates well the optimal solution. In particular our heuristic gives objective function values that are very close to the ones returned by CPLEX. GA is also remarkably close to the optimal solution, but is consistently outperformed by our heuristic and at the same time comes with significant processing overhead, as we shall show.

Random and FF placement, on the other hand, result in solutions that are increasingly far from the optimal as the number of available PMs increases, which is due to the fact that they do not take into account the main objectives and the respective policy. With an increase in the number of available PMs, the solution space also expands and these algorithms fail to "explore" it effectively. *Random* tends to result in solutions which utilize increasing numbers of PMs thus driving cost up, while *FF* tends to consolidate VMs in the least number of PMs, which reduces availability.

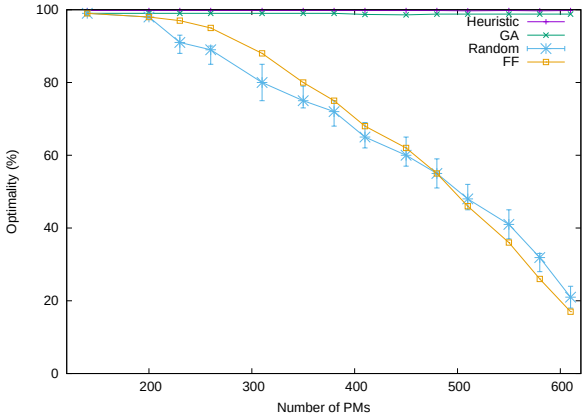


FIGURE 4.5 : Performance of different placement schemes for the same policy and increasing numbers of available PMs.

Then, we carry out an experiment where we fix the number of available PMs and measure how each of the candidate algorithms approaches the optimal solution for different policies. FIGURE 4.6 confirms that our heuristic tops the candidate schemes in terms of optimality. Importantly, policy-unaware schemes cannot perform consistently across the spectrum of available policies. For configurations with many available PMs, such as the one in this experiment, and when only availability is critical ($w_c \rightarrow 0$), *Random* may have acceptable performance since it will tend to distribute VMs uniformly across PMs. This advantage quickly diminishes as we move towards more cost-centered policies. Conversely, *FF* performs near-optimally when availability is not a concern and cost minimization is what mainly matters ($w_c > 0.7$), since it aims to pack VMs in as few PMs as possible. On the other hand, our two-step heuristic and our Genetic Algorithm, which incorporate policies in their design,

address successfully the cost-availability tradeoff, producing solutions near the optimal for all weight combinations.

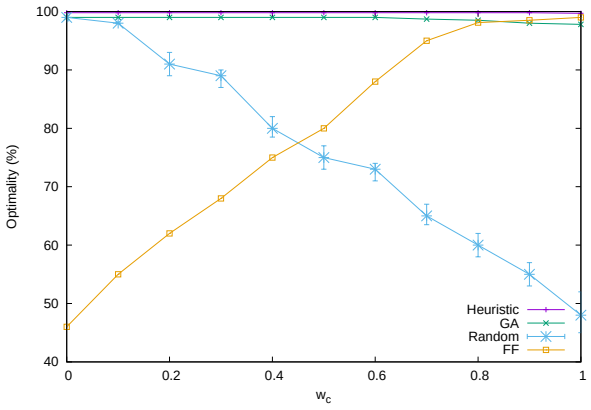


FIGURE 4.6 : Performance of different placement schemes for the same configuration (number of available PMs) as a function of the applied policy (criteria weights).

If we delve further into the performance of these two algorithms and consider the two objectives separately, we notice that TRAP achieves better performance in terms of both cost and availability at the same time, compared with the Genetic Algorithm. FIGURE 4.7 presents the (normalized) value of the cost component for different policies. As optimizing for cost becomes more important, the performance improvement of TRAP compared to the GA grows. The solutions derived using this scheme also improve on availability, especially when the applied policy seeks for a balance between the two objectives, as FIGURE 4.8 indicates. The gains in terms of optimality that our heuristic brings about compared to our GA can be attributed, to an extent, to the fact that the latter creates the original pool of solutions by randomly assigning VMs to PMs. In some cases, apart from slower convergence, this may affect the quality of the solutions produced by the GA.

Although both our heuristic and our GA approximate well the optimal solution, they come with different processing overheads. To quantify their running time performance, we carried out an experiment where we fixed the policy to $w_c = w_a = 0.5$ and measured the execution time of the CPLEX solver, the TRAP algorithm, and the GA. Regarding the GA, and as in our previous experiments, we configure the size of the solution pool to $S = 70$ and the number of generations to $G = 20$. These numbers were selected experimentally and represent a good compromise between execution time and the quality of the derived solutions. For reasons of a fair comparison, the execution time reported for the GA also includes the time it takes to determine the number of VMs to launch, for which we apply the first step of our heuristic; the GA then proceeds by selecting their appropriate placement. Our experiments

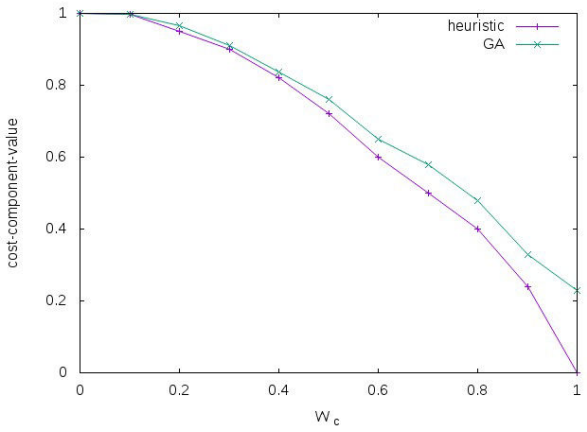


FIGURE 4.7 : Comparison between our two-step heuristic (TRAP) and our GA in terms of the cost objective for different policies (lower is better).

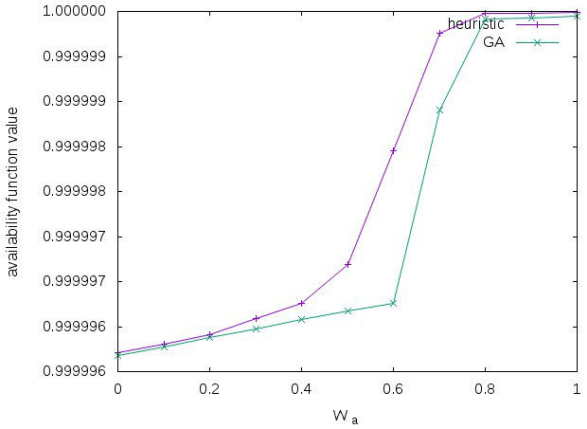


FIGURE 4.8 : Comparison between our two-step heuristic (TRAP) and our GA in terms of the availability objective for different policies (higher is better).

were performed on an Intel i7 machine with 8 CPU cores and 8 GB of RAM, running Ubuntu 14.04.

In the final experiments, we want to show the evolution of the execution time of the GA for different pool sizes and different numbers of generations, and then we evaluate the associated performance penalties.

FIGURE 4.9 and FIGURE 4.10 represent, respectively, the execution time as a function of the number of generations (G) and the pool size (S) . As expected, also taking into account the complexity analysis presented in Section 4.7, the running time of the GA increases significantly with G and S. FIGURE 4.11 and FIGURE 4.12 show the effects of the pool size and the number of generations on the quality of the solution. As can be seen from both figures,

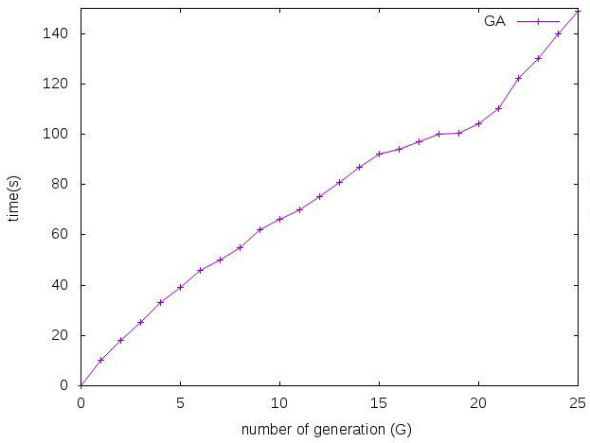


FIGURE 4.9 : Execution time of the GA as a function of the number of generations (G).

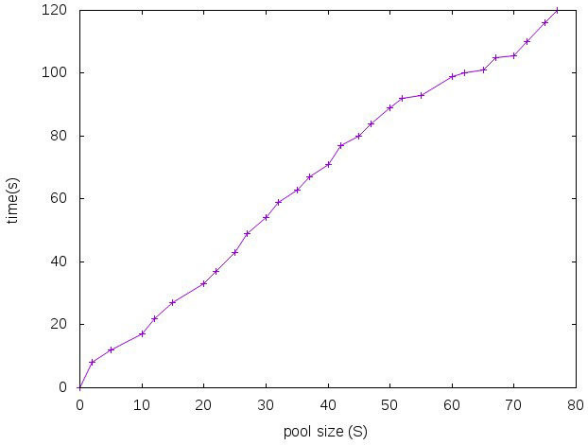


FIGURE 4.10 : Execution time of our GA as a function of the pool size (S).

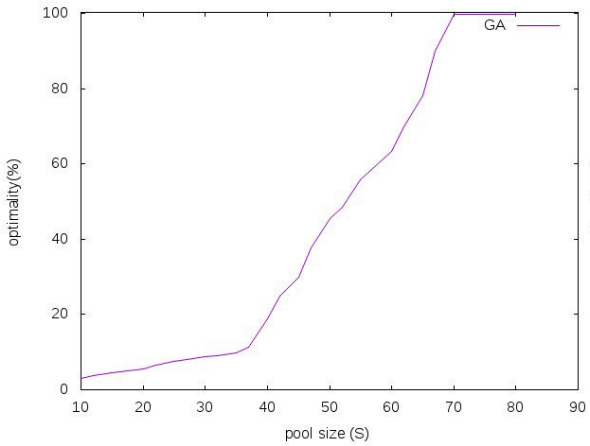


FIGURE 4.11 : Performance of the GA as the size of the pool (S) increases.

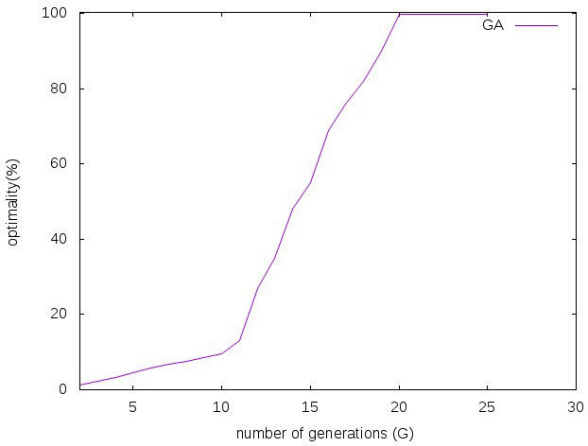


FIGURE 4.12 : Performance of the GA as the number of generations (G) increases.

there is no need to go beyond 20 for the number of generations and 70 for the pool size, since beyond these values the GA returns equivalent solutions close to the optimal, as derived by CPLEX, but for longer execution times.

To obtain the result of FIGURE 4.11, which represents the solution quality ("optimality") as a function of the pool size, we fix the number of generations to 20 and vary the pool size from 10 to approximatively 90 individuals. As can be seen, the solution approximates the optimal as the size of the pool approaches 70, which corresponds to the size we chose for all our experiments presented in our manuscript. This pool size represents a good compromise between execution time and the quality of the solution. Beyond this value of S, the result remains practically the same as the algorithm converges.

In a similar spirit, in FIGURE 4.12 we obtain the results by varying the number of generations between 2 and 30 and fixing the pool size to 70. As for the number of individuals (S), we obtain a good solution when the number of generations is greater than or equal to $G = 20$. This confirms our previous experiments where we chose $G = 20$.

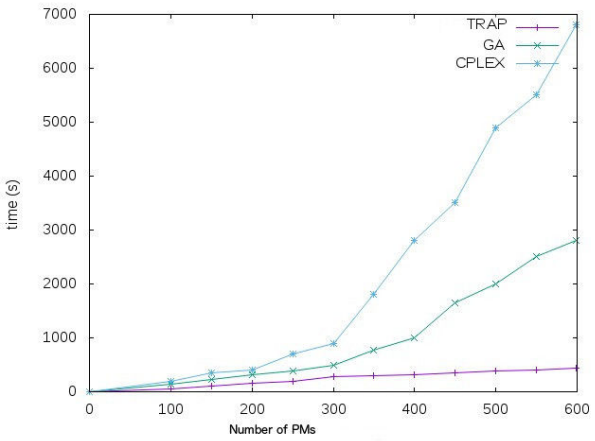


FIGURE 4.13 : Execution time as a function of the number of available PMs for different algorithms.

FIGURE 4.13 verifies that our TRAP scales well as the number of PMs grows. Even for 600 PMs, it produces a solution which is close to the optimal in the order of few minutes. The GA scales poorly for such problem instances, which is justified by its higher computational complexity (see Section 4.7). Its running time performance could be improved by reducing the number of generations or the solution pool size, but this would come at the expense of the quality of solutions it produces, which are already suboptimal to the ones of TRAP. Note that for more than 600 PMs, it was not tractable to derive the exact optimal solution using

CPLEX, since its execution time increases dramatically when the number of variables and/or constraints of the model increases.

4.9 A note on the relevance with ETSI NFV

Our model and algorithms reflect the particular characteristics of the vCDN use case described by ETSI [79]. From a service viewpoint, the ETSI vCDN use case specifically targets video distribution; our overall design is cantered around video delivery, too. The scenario described in the aforementioned specification involves a centralized CDN controller and a number of cache VNF instances distributed across data centers. These caches are identical from a functional perspective. This is in line with our model, where we are interested in allocating CPU resources to identical VNF instances following specific criteria, constraints and performance targets. We are focusing on the placement of cache VNF instances, as these are the ones which will mainly absorb the load of the vCDN, not considering in our algorithms the placement of other components of a vCDN instance. Caches are relatively simple network functions and, as such, it is reasonable to assume that there is a 1:1 mapping between the cache instance and the VM that hosts it, treating it as a VNF composed of a single VNF component. This is the reason why in this thesis we use the terms VM and VNF instance interchangeably.

Furthermore, at the heart of our model is the cost-availability tradeoff, with both criteria appearing in the objective function and the constraints. This makes our model particularly relevant to NFV. The ETSI has published specifications on NFV resiliency requirements [78], explicitly mentioning the issues and tradeoffs we face and try to capture with our model. The relevance of our work with NFV from a modelling perspective is further established by the fact that, as per the ETSI specifications, there is a requirement for service availability to be defined on a service basis and to be considered together with the total cost and its importance for the customers (in our case content providers). The policy mechanism that we have introduced serves exactly to this end. Different policies and availability constraints can be thus put in effect for different service instances.

Finally, although the focus of this work is on CDN, other types of services are not out of the question. We remark that our model and algorithms could be extended with minor adaptations towards other families of virtualized applications, provided that a similar cost and availability model are assumed, and that the VNF instances to place have identical functionality, as is the case for CDN caches. Such potential extensions are a topic for further study.

4.10 A note on network resource allocation

Our model and algorithms are oriented towards computing resources; allocating network resources is not in its scope. A basic underlying assumption of this work is that for a given level of user demand for video traffic (which has been translated to specific compute resources), it is more straightforward for the operator of the CDNaaS infrastructure to calculate the amount of necessary network resources. Then, it can accordingly provision the network paths between the virtual instances and end users, and/or perform the necessary traffic engineering tasks.

We should also remark that with appropriate load balancing policies in place (this is straightforward to implement at the CDN service level), a VM with more vCPUs assigned would receive more video requests. Therefore, the traffic that a VM (and in turn a PM) receives is proportional to the amount of compute resources allocated to it. This means that processing resources can be directly linked/translated to networking ones. The system operator then has to provide as input to the algorithm the number of CPU resources per host for which it can guarantee the availability of network capacity necessary for the traffic these vCPUs will handle, which as we argue here is realistic to assume.

Therefore, in this scenario where CPU and network capacities are tightly coupled, our solutions indirectly capture network constraints. However, this would not necessarily hold true if network parameters were part of the objective function to optimize. For example, if network link reliabilities need to be considered in the service availability model or network link costs need to be accounted for, our model and algorithms would have to be reconsidered.

4.11 Conclusion

This chapter presented the use of our quantitative results to derive resource allocation/service placement algorithms that aim to optimally satisfy customer demand, minimize on cost and offer availability guarantees. We studied the problem of jointly allocating compute resources to virtual instances and their placement on an NFVI for the provision of CDN-as-a-Service. Our focus was on providing the optimal number of vCPU then simultaneously addressing the conflicting requirements for improved service availability and reduced management cost. To this end, we first proposed a measurement-driven algorithm for determining the optimal number of vCPU resources. Second we proposed a multi-objective optimization formulation of the problem, as well as efficient heuristic algorithms to solve it. We demonstrated by simulation on a real PoP topology the importance of QoE-awareness in effectively addressing the tradeoff between service quality and cost. We also showed quantitatively how our algorithms optimally address the cost-availability tradeoff. By comparing our

scheme to simple baseline algorithms that are often used as benchmarks in related work, we demonstrated that in order to address more effectively both objectives, a flexible scheme which incorporates their relative importance in its design is necessary. Our approach can prove beneficial for the operator of such a system in order to appropriately dimension a virtualized CDN service, implementing resource allocation policies that reflect customer (i.e., content provider) preferences, and assisting in the establishment and enforcement of specific service-level agreements. The heuristic algorithms we devised run efficiently, without sacrificing on solution quality. We have shown experimentally that they can derive near-optimal solutions in the order of few minutes for large problem instances, where exact solutions by means of solvers such as CPLEX are intractable to get. Finally, we explore alternative strategies to tackle our problem by proposing a Genetic Algorithm suitable for our model, inspired by the related state-of-the-art. Our heuristics were shown to outperform the Genetic Algorithm, both in terms of optimality and efficiency.

Although the proposed scheme allows deriving a near optimal solution to place vCDN resources over a telco or federated cloud, it requires a special focus when fixing the weight values, which mainly rely on the vCDN service type to deploy. For example, for a vCDN that delivers highly-popular content, higher reliability may be required by the content provider, and thus w_a should be significantly higher than w_c. For deployments which deliver less popular content or which involve short service duration, on the other hand, the customer might be less interested in high availability guarantees. In these cases, the operator might select a cost-centered policy with a high value for w_c to reduce operational cost and possibly to be able to provide a more affordable offer to the customer. Therefore, the service type has critical impact on the selected deployment policy.

A line of research studies detailed in the next chapter is the extension of our design towards edge computing. The main challenges therein lay in the inherent scarcity of edge resources, and the heterogeneity naturally introduced in the underlying physical infrastructure both in terms of operating cost and reliability. On the other hand, edge computing offers the potential for lower latency and thus improved user experience while saving on the operator's backhaul network capacity. These new features require modifications both at our model and our algorithms which are the subject of the next chapter.

CHAPTER 5

ULTRA-RELIABLE AND LOW LATENCY VNFS PLACEMENT IN A MEC-NFV ENVIRONMENT

5.1 Introduction

Ultra-Reliable Low-Latency Communications (uRLLC) services, such as connected car, industry 4.0 and critical IoT, are expected to be deployed using the upcoming 5G systems. uRLLC services require very low latency access between the remote server and the clients, while running in a highly reliable environment to guarantee service continuity. While 5G introduces novel mechanisms to ensure low latency, via MAC layer scheduling and network slicing [54], it requires running the servers belonging to uRLLC at the cloud edge to maintain a low end-to-end latency. Indeed, Edge Computing allows the deployment of applications close to end users to reduce access latency, as compared to the central cloud access. Two technologies emerged to enable the edge cloud, FoG and MEC. The former is expected to run at the edge router, while the latter, as mentioned in the context chapter, supported by the ETSI as well as network operators, is expected to run close to 4G/5G base stations (i.e., eNodeBs). As explained in chapter 2 several ETSI documents have been issued [73, 74] to define a reference architecture and functional blocks to enable running MEC applications at the edge. Moreover, ETSI has recently started working on updating the MEC reference architecture, initially introduced in chapter 2, to run MEC entities in a NFV environment [72], to take advantage of the latter's mature deployment environment in terms of implementation and enabling tools, such as the Open Source Mano (OSM)[1] and OpenBaton[2] orchestration suites. The NFV orchestration process, known as the NFVO (presented in chapter 2), is in charge of the automation of Network Service (NS) deployment over the virtualized infrastructure, which mainly consists in the instantiation, placement, and life-cycle management of VNFs composing a NS. By integrating MEC in NFV, ETSI considers MEC applications as classical VNFs, hence using the same orchestration and management process at a cen-

1. https://osm.etsi.org
2. https://openbaton.github.io/

tralized NFVO. It is important to remind that MEC applications need to be hosted at the edge aiming at guaranteeing low latency access to the service. However, most of the used placement algorithms in NFV such as our algorithm previously presented, aim to reduce deployment costs, increase the users' QoS, ensure service availability, etc., ignoring MEC applications' constraints in terms of low latency. This may MEC application instances to be run at central clouds, as the NFVO and its related placement algorithms do not differentiate between MEC applications and other VNFs.

After defining the MEC-NFV architecture in the context chapter, we fill this gap by proposing a placement algorithm tailored to uRLLC services in the context of MEC in NFV. First, the proposed scheme classifies all cloud resources (central and edge cloud) according to their access latency to the user data plane, which avoids the need for additional information to differentiate between edge and central cloud resources, thus being in compliance with the MEC in NFV concept. Then, the placement problem is formulated as an optimization one, where the objectives are to minimize latency and to maximize service availability (reliability); these correspond to the main deployment criteria for uRLLC services.

This chapter is organized as follows. In Section 5.2 we present related work on VNF placement in different environments. We propose an adaptation of our formulation presented in the previous chapter, by addressing this time, the VNF placement problem for uRLLC in Section 5.3 by integrating the latency function. We provide solutions to this problem by adjusting our Genetic Algorithm (Section 5.4), but also by using the CPLEX solver as a benchmark. Our results are shown in Section 5.5, before we conclude in Section 5.6.

5.2 State of the art

The VNF placement problem in Telco or federated clouds has recently received significant attention. These types of cloud involve distributed data centers (DCs) connected into a common resource pool to deliver different cloud services.

The VNF placement problem is similar to the VM placement problem, as VNFs are composed of virtual instances (VMs or containers) that execute network functions. As highlighted in chapter 4, a suitable VNF placement is mostly motivated by maximizing resource utilization or overall performance, while minimizing cost in terms of energy consumption, network traffic or penalties associated with SLA violations under various constraints [20, 91]. Although these metrics are well suited for NFV, putting MEC in the picture may require other metrics to be considered, such as latency. Indeed, one of the advantages of placing VNFs at the edge is the proximity to end-users, which leads to a low-latency service deployment as reported

in [101]. Specifically the authors analyze the MEC reference architecture and deployment scenarios, which offer multi-tenancy support for application developers, content providers, and third parties. They highlight the important changes towards the vision of 5G where MEC is known as one of the key emerging technologies, thanks to its significant contribution to low latency assurance. Therefore, placement algorithms should be updated to integrate latency as a criterion, in addition to other metrics.

Only few works have addressed the problem of VNF placement in a mixed environment that includes edge and central clouds [50, 5]. Generally, VNFs are placed either in physical machines within central cloud infrastructures only [4], or in MEC servers [102, 52].

Other works [103, 9] investigate the placement in the context of a hybrid federated cloud, using a combination of different cloud infrastructures, without however involving the edge. In this context, the authors in [5] also propose a VNF placement solution that captures the trade-off between cost efficiency and QoE in a multi-cloud environment.

Similarly to our work, some recent approaches [50, 2] outstrip the common cost and resource optimization by introducing more specific VNF placement and provisioning optimization strategies over edge and central cloud infrastructures. Jemaa et al. [50] take into account QoS requirements such as minimum bandwidth and maximum end-to-end latency. In particular, their objective is to minimize the maximum utilization of the edge cloud by deploying more VNFs in the traditional cloud.

In the previous chapter, we studied the problem of jointly allocating compute resources to virtual instances and their placement on a traditional cloud infrastructure for the provision of CDN-as-a-Service. Our focus was on simultaneously addressing the conflicting requirements for improving service availability and reducing management cost.

The proposed solution in this chapter is similar in spirit but is put in a different context. In particular, as in the NFV environment, we provide a multi-objective optimization problem formulation and apply the service availability model we introduced in chapter 4. However, from a system model perspective, this part of the work has the following distinct differences: (i) physical machines are not considered identical regarding their reliability and the cost of deploying VMs on top of them, thus differentiating between edge and central cloud hosts, (ii) latency minimization is one of the objectives, (iii) the CPU resource requirements per VNF instance to deploy are part of the problem input. For these reasons (differences) we redefine our models by adapting them to the uRLLC requirement in the next section.

5.3 Problem formulation

We assume that an operator manages a set of data centers (DCs), which can either be installed at the network edge, and therefore close to end-users, or in the traditional central cloud. The uRLLC service is composed by a number of VNFs, which are in turn composed of a set of VMs (a VNF could be composed of one or more VMs to guarantee scalability). The operator aims to derive a suitable placement of those virtual instances (VMs) in DCs, where each DC has a set of physical machines (PMs) where VMs can be launched according to the demand. The objective is to find an appropriate placement of the n VM composing a service instance on a subset of the m available PMs. This is translated in our system as a binary assignment matrix $X = (x_{ij})$ with $i \in [1, n]$ and $j \in [1, m]$, where x_{ij} is equal to 1 if VM_i is hosted at PM_j, and 0 otherwise. The number of VMs to instantiate, as well as the CPU requirements of each VM, are part of the problem input. The calculation of the optimal assignment should minimize the average access latency of the service deployment, consider the CPU capacities of the underlying physical hosts, not exceed a budget, and respect a service availability constraint. We recall that we target a uRLLC service, which requires both low-latency access and reliability.

5.3.1 Latency model

As initially mentioned, in this chapter, we consider two types of hosts (PMs): ones which are located at centralized clouds/NFV infrastructures, and ones at MEC DCs. Hosts are grouped by DC and all hosts belonging to the same DC are assumed to share the same access latency value. Note that by characterizing DCs/hosts by latency, we do not need to explicitly differentiate between MEC and central cloud PMs; for a delay-minimizing algorithm, only the latency property is relevant.

We define access latency as the latency for users to access the uRLLC service. $D = [d_1\ d_2\ \dots\ d_m]^T$ is a vector where d_j is the latency characterizing the $j-$th PM and m is the total number of available PMs across all DCs. A specific assignment X is characterized by a n-dimension latency vector $D' = [d'_1\ d'_2\ \dots\ d'_n]^T$, whose value depends on the PMs that are utilized to host the n VMs composing the service, and which we formulate as

$$D' = X \times D.$$

The element $d'_i = \sum_{j=1}^{m} x_{ij} d_j$ corresponds to the latency of the PM which is hosting the $i-$th VM (since a VM is placed at exactly one PM, only a single term in the above sum will be non-zero).

We define the latency of an assignment X as the average of the elements of D', i.e., the average latency over all deployed VMs:

$$L(X) = \frac{\sum_{i=1}^{n} d'_i}{n} \tag{5.1}$$

5.3.2 Availability model

As in the previous chapter, in our model, we define the availability as the ability of the system to offer at least a minimal functional and accessible service. In particular, a service instance is considered available if at least one of its constituent VMs remains accessible, which implies that the PM hosting it is accessible as well. In this part of the work, we apply the same availability model formulation, initially defined in section 4.5.3.

We remind and adapt the following assumptions:

- We consider two types of PMs, central and edge cloud ones.
- The probability of failure of a VM i is $q_i^{(V)}$, independently of the other VMs and PMs, and irrespectively of the load imposed on the VM.
- The probability of failure of a PM j is $q_j^{(P)}$, independently of the other PMs or the load imposed on it. It is reasonable to assume that at the edge, hosts have a greater failure probability and are thus less reliable. Indeed, VM replication or migration inside the edge is harder to ensure due to the scarcity of resources compared to the central cloud.
- If a PM fails, all VMs deployed on it fail because of that.
- A VM may become inaccessible either because it fails or because the PM that hosts it fails.
- The system operator knows the value of the PM and VM failure probabilities, as a result of measurement studies, prior experience, or other historical information.

Since VM failures can be correlated due to their dependence on the underlying PMs, we define a *correlated group* of VMs as the set of VMs that are executed on the same PM. A correlated group is available under the following conditions:

- The PM is up.
- At least one of the VMs deployed on the PM does not fail.

As previously stated, at least one correlated group should be available in order to have a service/application deployment available. Since correlated groups fail independently, the probability that a service deployment is available is given by

$$A(X) = 1 - \prod_{j \in [1,m] | \Sigma_{i=1}^{n} x_{ij} > 0} \left[q_j^{(P)} + (1 - q_j^{(P)}) \prod_{i \in [1,n] | x_{ij} = 1} q_i^{(V)} \right] \tag{5.2}$$

We should note here also that, by construction, any feasible solution includes at least one PM with at least one VM assigned to it; therefore, both product terms in (**??**) are over non-empty sets.

5.3.3 Cost model

In the previous chapter, we considered that the deployment of a VM comes with a fixed management overhead. In this part of the thesis, however, not all VMs come with the same cost. In particular, we treat cost as a function of the associated VM workload (in terms of the number of vCPUs allocated to the VM and under the assumption that more powerful VMs handle more application workload). This cost may also account for the energy consumed for booting the VM or for operating other system- or service-level components. Therefore, we define a VM capacity requirement vector $P = [p_1\ p_2\ \ldots\ p_n]$, which represents the CPU capacity needed for the allocation of each VM; p_i is the service/application requirement in terms of the number of virtual CPUs for VM_i.

We also assume a PM-level cost, which is a fixed overhead and is not a function of the PM workload nor the number of VMs hosted by it (e.g., energy cost for keeping the physical machine in an operating state).

Therefore, the cost of an assignment X at the VM level is given by

$$C_V(X) = e_V \sum_{i=1}^{n} p_i, \tag{5.3}$$

where e_V is the fixed cost per vCPU assigned to a VM.

The cost at the PM level is determined by the number and the type of the PMs involved. Note that we consider it more costly to operate on edge hosts. This is due to the value and scarcity of edge resources. We assume that when involving a PM in an assignment, this comes with a fixed PM-level cost. We encode these fixed costs in vector $K = [k_1\ k_2\ \ldots\ k_m]$, each element of which corresponds to the cost associated with a specific PM. The PM-level

cost of an assignment is defined as the sum of the fixed costs of all the PMs that are used, i.e., the PMs that host at least one of the service's VMs:

$$C_P(X) = \sum_{j=1}^{m} k_j \mathbb{1}(\sum_{i=1}^{n} x_{ij} > 0). \tag{5.4}$$

The overall cost of a placement is then given by

$$C(X) = C_V(X) + C_P(X). \tag{5.5}$$

5.3.4 Objective

The objective of the system is to derive an optimal assignment that minimizes latency while maximizing availability. Each criterion drives VM placement towards a different type of solutions. Focusing on a low latency service deployment will lead the system to place the service components at edge servers, which provide shorter response times but are more expensive. On the contrary, traditional central DCs offer higher availability but with longer delays. Since it is not possible to optimize for both criteria at the same time, we apply the same method used in the Cost- and availability aware VNFs placement in CDNs, a scalarization approach to transform the problem to a single-objective one. The relative importance of the two criteria in deriving an optimal placement is dictated by a specific *policy*, which is encoded in a pair of weights (as in the previous chapter). However in this part the weights are w_l for latency and w_a for availability, such that $w_l + w_a = 1$. Given a specific policy that depends on the particular uRLLC service, the system operator derives the optimal solution to the following problem:

$$\begin{aligned}
\underset{X}{\text{minimize}} \quad & w_l L(X) - w_a A(X) && (5.6)\\
\text{subject to} \quad & C(X) < E && (5.7)\\
& A(X) > A && (5.8)\\
& \sum_{j=1}^{m} \mathbb{1}(x_{ij} > 0) \leq 1, \forall i \in [1, n] && (5.9)\\
& \sum_{i=1}^{n} p_i \times x_{ij} \leq u_j, \forall j \in [1, m]\,, && (5.10)
\end{aligned}$$

with $U = \{u_1, u_2 ... u_m\}$ representing the available CPU capacity for each of the m PMs.

In the proposed formulation we consider a budget constraint (5.7) which limits the cost of the service deployment to below E and ensures that an initial budget agreement with

the customer is not violated. Constraint (5.8) ensures a service with a minimum availability. Constraint (5.9) ensures that each VM is assigned to a single host, since a VM cannot be split, and constraint (5.10) guarantees that the capacity of each host is not exceeded.

5.4 A Genetic Algorithm for VM placement to guarantee an uRLLC system

Many VM or network function placement problems are known to be NP-hard [66] and various heuristics are being proposed for this reason. Therefore, we remind in this section the GA *meta-heuristic* initially presented in the previous chapter as an alternative solution. We adapt the GA tailored to our model to tackle our VM placement problem. As mentioned GA solves an optimization problem applying iteratively two main operations: *crossover* and *mutation*. It encodes each potential solution as a chromosome (a sequence of characters) of properties called genes and characterizes each solution with a fitness value, which is an expression of the solution's quality. The GA starts with an initial pool of candidate solutions and iteratively tunes them to produce new generations of better quality by using *crossover* to generate new chromosomes from selected parents and *mutation* to randomly adjust a chromosome.

In this part of the work, our GGA integrates our availability and latency objectives. Our fitness function is dictated by the weights (policy) and is a direct application of th objective function equation (5.6).

Our GGA takes as input the number of VMs and creates an initial solution by generating S random assignments of VMs to PMs. For each solution in S, each group of VMs on a PM is a gene. Once the initial solution pool is generated, the GGA runs for G generations.

We remind that the *ranking-crossover* operator is an improved version of the classical crossover. The latter choses genes blindly from selected parents. On the other hand, *ranking-crossover* selects the genes which have a higher rank according to the average value of specific *efficiency* functions defined per objective, hence expected to produce higher quality offspring.

In our case, for each *ranking-crossover* operation, our GGA calculates the efficiency value for each involved gene from two random chromosomes.

We chose as an efficiency function the distance between a gene's derived values for latency and availability given by (5.1) and (5.2) respectively, and ideal ones. These ideal values are probably unattainable at the same time by a single gene and correspond to a *utopian* case. For example, we have set the ideal values to $u_l \approx 0$ (latency) and $u_a = 100\%$ (availability). Our efficiency function is the adaptation of the *Euclidean distance* between the utopian point (u_l, u_a), and the point representing the gene's latency and availability values (l, a). The efficiency function adapted the the uRLLC is formulated as:

$$\mathcal{E} = \sqrt{(u_l - l)^2 + (u_a - a)^2}.$$

After deriving $\mathcal{E}$ for each gene, the GGA ranks them from the smallest distance, which corresponds to the highest rank, to the largest one, and creates a new chromosome by combining the highest-ranking genes.

Note that for each generation G, this procedure is repeated in order for a number of new chromosomes to be generated, which is specified by the crossover rate r_c parameter. Therefore, the GGA produces $r_c S$ offspring on average on each generation. The solution pool population at the end of a generation is constructed by evaluating the fitness function for each chromosome and keeping the top-S of them. After G generations, our algorithm terminates by returning solution with the highest fitness function value. During the whole process our GGA eliminates solutions that violate any of the constraints.

The overall complexity of this GGA adaptation equals the complexity calculated in section 4.7 which is: $\mathcal{O}(Sn + GSmlogm)$.

5.5 Numerical results

For our experiments, we consider different configurations, where we use different types of hosts, central cloud and MEC ones. We show the positive impact of using edge hosts in deployment delays and how we capture the trade-off between latency and availability. We also compare our scheme with a CPLEX[3] implementation in terms of the quality of the solution and execution time.

We assume a uRLLC service composed of 90 VMs that need to be deployed over a federated cloud (including edge DCs). We consider a system with 150 PMs, 100 PMs within the traditional cloud each with a capacity between 5 and 15 vCPUs, and 50 MEC servers with a capacity between 1 and 6 vCPUs each. Host capacities are selected uniformly at random.

3. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

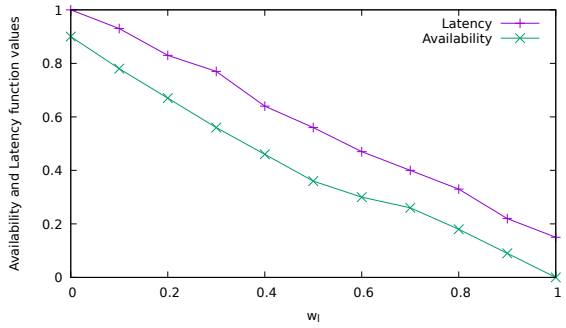


FIGURE 5.1 : Availability and latency as functions of the selected policy.

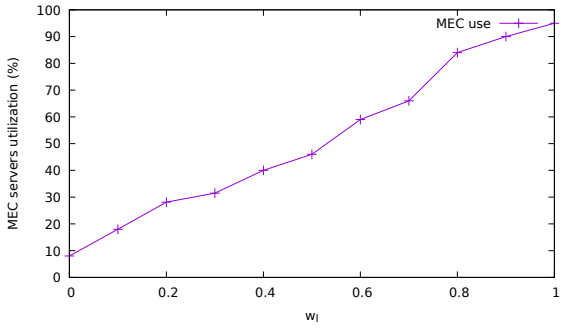


FIGURE 5.2 : MEC servers utilization as a function of the policy.

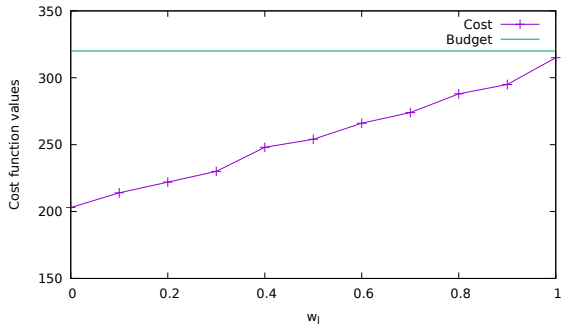


FIGURE 5.3 : Evolution of cost with the latency-oriented policy. The straight line represents the cost constraint (budget).

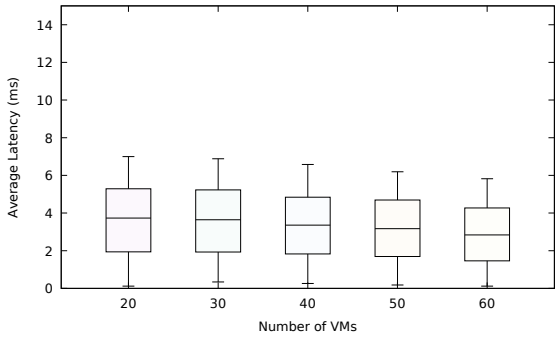


FIGURE 5.4 : Box-plot showing VMs' impact on the average latency of a service deployment.

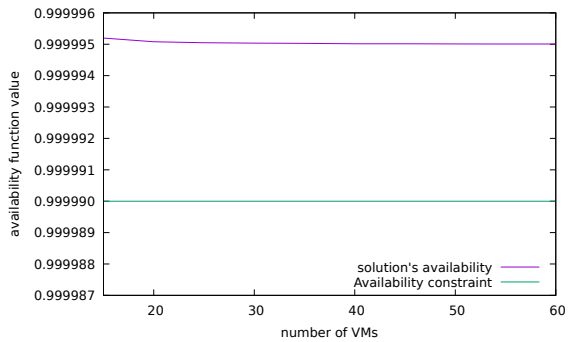


FIGURE 5.5 : Availability as a function of the number of VMs. The straight line represents the availability constraint.

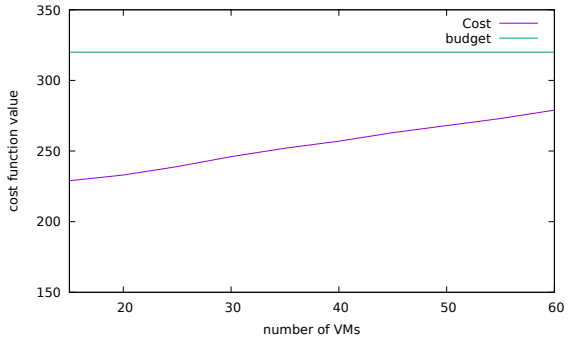


FIGURE 5.6 : Cost objective value as a function of the number of VMs. The straight line represents the cost constraint.

We fix the budget to $E = 320$ and assume that all PMs of a specific type incur the same fixed cost. For each edge server, the PM-level cost is $k_e = 2$, while for each central cloud host it is $k_c = 1$. The cost of one vCPU assigned to a VM is $e_v = 1$. All the VMs have the same failure probability $q_V = 0.001$. The PM failure probabilities are set to $q_{P1} = 0.004$ for MEC servers and $q_{P2} = 0.002$ for central cloud servers, since we assume that the reliabilities are different for the two types of hosts. We select uniformly at random the delays, which we set to between 1 and 5 ms for the 50 edge servers, and between 4 and 10 ms for the 100 central cloud servers.

We run both algorithms (CPLEX and GGA) for: (i) the same number of vCPUs assigned to each VMs, (ii) the same number of VMs (unless otherwise specified), and (iii) the same PM delays (D). Regarding the GGA we configure the solution pool size to $S = 80$ and the number of generations to $G = 30$. We chose those values experimentally, since we found them representative to a good compromise between execution time and quality of the solutions.

FIGURE 5.1 outlines the availability and latency function values for our algorithm for different policies. The values are normalized by mapping the lowest and highest value per objective to 0 and 1 respectively. As it is expected, the results indicate that the more the policy is latency-oriented ($w_l > 0.6$), the more the algorithm sacrifices on availability. In particular, since our GGA is guided by the applied policy, the objective function value of equation (5.6) decreases because the chosen servers for the deployment have small delays (small values of d_j); more edge servers are used to deploy the service components than central ones.

This is more clear in FIGURE 5.2 that represents the usage rate of MEC servers as a function of the applied policy. It can be seen that when w_l increases, more hosts within the edge are used, because the delays on the edge are smaller and our algorithm honors hosts which offer minimal latencies.

FIGURE 5.3 presents the evolution of cost during the placement, where the y-axis represents the values of the cost as a function of the given policy depicted in the x-axis. The straight line represents the maximum budget set to $E = 320$ (which can be fixed with the customer). As can be seen, the cost increases substantially with w_l. We argue this by the fact that the GGA uses more MEC hosts for the VM placement, which are by definition more expensive.

In our second experiment we run our GGA by fixing the policy to $(w_a, w_l) = (0.5, 0.5)$ (uRLLC service that requires both latency and availability), using the same number of PMs and their capacities, and for the same number of vCPUs (200). We vary the number of VMs in an interval of $n \in [0 - 60]$ for 100 iterations and present results with 95% confidence intervals.

The results of the latter experiment are shown in FIGURE 5.4 as box-plots illustrating the evolution of the average latency according to the number of VMs. As depicted in FIGURE 5.4, when n increases, the average latency decreases. In fact, adding more VMs leads to their placement in smaller DCs, which in our case corresponds to edge PMs; the more VMs we have, the less vCPUs are allocated to each one of them since the number of vCPUs remains the same for all our tests.

In FIGURE 5.5, we highlight the evolution of availability as a function of the number of VMs, for the solutions obtained by our GGA. The straight line represents the availability constraint, set to class five (99,999%) of reliability requirement for uRLLC services.

From the resulting plot we can see that the availability slightly decreases with the number of VMs, which is compatible with the growing use of MEC servers previously pointed out. Considering the small capacities within the edge servers, the VMs with the small number of vCPUs assigned to them are placed on those servers. However, even if the values of availability decrease, the availability constraint is not violated.

Regarding the cost, FIGURE 5.6 illustrates its evolution as a function of the number of VMs for the same case. We clearly see that the cost increases with the number of VMs, which positively supports our results from the two previous figures. The cost is affected by the use of PMs from the edge, which are more expensive.

In our third experiment, we implement our model in the CPLEX environment and use its optimizer to derive exact optimal solutions. We compare and evaluate the CPLEX results with our GGA.

FIGURE 5.7 presents a comparison of the objective function values obtained for the same configuration for our GGA (green curve, “x” points) and CPLEX (purple, cross points) as a function of the applied policy. Since we are solving a minimization problem, lower values for a specific weight combination mean better performance. Our meta-heuristic is shown to approximate well the optimal solutions provided by CPLEX.

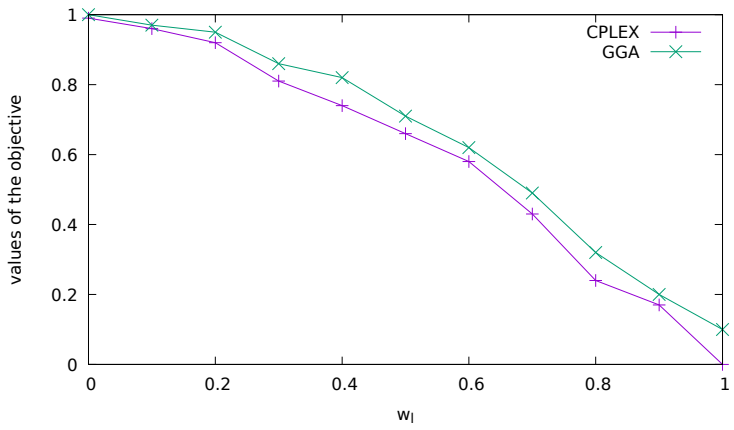


FIGURE 5.7 : Comparison between our GGA and CPLEX in terms of the objective for a specific policy.

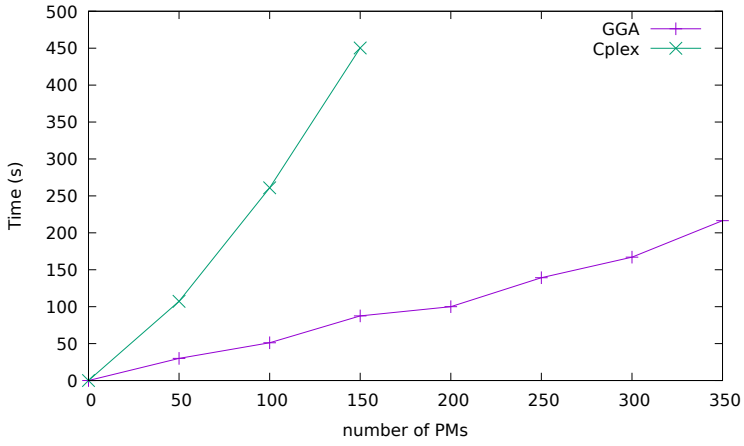


FIGURE 5.8 : Execution time as a function of the number of PMs for our GGA and CPLEX.

FIGURE 5.8 presents the execution time results for both CPLEX and GGA for increasing numbers of available servers. Our experiments were performed on an Intel i7 machine with 8 CPU cores and 8 GB of RAM, running Ubuntu 14.04. It can be seen that our GGA scales well as the number of PMs grows. Even for more than 200, it produces a solution close to the optimal in an order of a couple of minutes. On the other hand the execution time of CPLEX increases dramatically, and it stops producing solutions for problem instances with more than approximatively 150 PMs. Note that we carried out those experiments by fixing the policy to $w_l = w_a = 0.5$.

5.6 Conclusion

In this chapter, we presented our study on the problem of placing VNF instances on a mixed environment which considers two different types of hosts, namely PMs from the edge and the central cloud. We adapted our multi-objective optimization formulation of the problem that captures the trade-off between service access latency and availability, which correspond to the criteria of uRLLC services. To solve this placement problem, we provided an appropriate adaptation of the Genetic Algorithm meta-heuristic. Our experimental results showed that our solutions are close to the optimal, while simultaneously it takes considerably less time to derive them than an exact algorithm provided by CPLEX. We have also shown how our method can reduce delays and still provide a highly-available service.

Our approach tackles important aspects regarding uRLLC services, but more needs to be done. Future work and perspective in this direction are presented in the next chapter, where we will conclude this manuscript and highlight our perspectives.

CHAPTER 6

CONCLUSIONS AND PERSPECTIVES

Internet traffic is dominated by data distributed over CDN infrastructures. The ever-increasing user expectations at low cost make the delivery of high QoE and high user satisfaction for multimedia services more important than ever in the eyes of ISPs. In this thesis, we presented our architecture for CDNaaS provision. We introduced architectural and system aspects, and focused on creating the support for a multitude of resource allocation and virtual function placement mechanisms as architecture plugins. Then, multiple models and algorithms were proposed to solve the problem of VNF placement in two different cloud computing environments, namely, central and Edge cloud. Each model was targeting different objectives aiming to satisfy both the service providers and the end-user.

We believe that this manuscript tackles topical issues in both cloud computing in general and VNF placement in particular. However, this is just the beginning of these topics of interest, and more investigations need to be done. After concluding our manuscript, we present tracks on future works.

6.1 Results obtained during the thesis

The contributions of the thesis are:

- *An architecture for on-demand service deployment and provision over a telco CDN*: in chapter 3 we proposed and described our implementation of an architecture allowing a telecom operator to offer a virtual CDN service to content providers on demand. Our system offers a rich northbound customer API and allows for the application of sophisticated service placement algorithms, given the inherent awareness of the provider of its current network conditions and capacity, and its presence close to end users. Our approach improves on resources management and service provision flexibility compared to traditional telco CDNs. In order to get a better understanding of the potential of virtualization and containerization as key enabling technologies to support our CDNaaS vision, we have carried out extensive testbed experiments. These experiments helped us to address the issues of VNF placement and resource allocation.

- *Cost and Availability aware resource allocation and virtual function placement for CDNaaS provision*: In the second contribution we proposed a measurements-driven algorithm and we have shown by simulation the importance of QoE-awareness in effectively addressing the tradeoff between service quality and cost. Then we studied the problem of jointly allocating CPU resources to virtual instances and their placement on a cloud infrastructure for the provision of CDN-as-a-Service. We proposed a multi-objective optimization formulation to the problem as well as efficients heuristic and meta-heuristic algorithms to solve it. We evaluated our algorithms by comparing them with two simple baselines strategies (Random and first fit placement algorithms) and with the optimal CPLEX. Our results demonstrated how our propositions outperform the baselines and considerably approach the exact solution provided by CPLEX in significantly less time.

- *Latency and availability driven VNF placement in a MEC-NFV environment*: in the last contribution of this thesis, we propose a method to place VNFs instances in a two-tier environment namely MEC and NFV environment. We adapt our previous multi-objective optimization formulation of the problem to capture, this time, the trade-off between service access latency and availability. The latter corresponds to the criteria of uRLLC services. The idea was to propose a solution that reduces delays and still provides a highly-available service. We solve the proposed problem with an adaptation of the meta-heuristic the Genetic Algorithm. Our simulations showed that our solutions are close to the optimal while taking less time than the optimal CPLEX.

6.2 Perspectives

- *In the context of CdnaaS*: first our cost functions do not consider whether PMs are already active hosting virtual instances, the idea will be to consider the status of PMs. For example in our model we can add a binary variable equals to 1 when the PMs is busy or 0 otherwise; it should be noted that favoring such PMs in the placement process would lead to more significant energy savings.

  Second different types of components could be considered, such as caches, which may influence the type of resources needed and thus the number of vCPUs as well as the number of VMs. Third, our mechanism will take into account failure recovery in case of PM breakdowns, especially at the edge and for critical services, such as the connected car.

  Fourth, we give few examples in the following of how our models can be applied to satisfy Customer's request. Since no SP will accept an unavailable network functions,

if a cloud data center or an entire region loses service. VNFs must achieve the same strict objectives as network functions.

1. Quantifying the service features in terms of availability levels (from low to five nines), cost (from low to expensive) and the Quality of the transmitted video specification (QoE) (between $3.5 - 5$ MOS). These metrics can be discussed before the service deployment between the SP and the CP.
2. The costumer chose according to his needs and priorities a percentage of both availability and cost (the sum of them equals 100%). The customer also requests the targeted number of end-users (part of the demand) and fixes his budget.
3. Our Cost- and availability-aware placement algorithm inputs the percentage of availability, the quality and cost constraints and the desired number of end-users requested by the customer. Then the algorithm translates the percentage into values of weights, i.e: 90% of availability corresponds to $w_a = 0.9$ and 10% of cost to $w_c = 0.1$. Finally, the algorithm provides the placement strategy: the mapping VM-PM to integrate in the VNF placement. Thereby end-users are enjoying an acceptable QoE (chosen by the CP), and the CP benefits from high availability of the service at cost that he was aware of.

Note that these latter steps can also be applied to achieve an uRLLC service on MEC in NFV environment.

- *Pricing models and SLA design, monitoring and enforcement*: finally, we are exploring the use of our model with respect to the establishment of SLAs with the customer, in order to create a clear common understanding about different services. The idea is to derive pricing strategies for the offered vCDN service. As mentioned the operator can select specific points representing response times for a specific request percentile. This sustains larger workloads while meeting specific response time requirement, which can be encoded in an SLA; allocating more resources (e.g., launching a new VNF instance or scaling up an existing one with more vCPUs). Taking such information into account, the operator can offer different pricing schemes. We propose two examples:
  - *Flat-rate, non-scalable deployment*: This is a basic service which provides no elasticity. The price is set based on an initial demand estimate that is provided by the customer (content provider): On service instantiation, the customer can express a target number of end-user HTTP sessions per region, with a specific objective with respect to response times. The service provider then decides on the amount of resources (HTTP server VNF instances and the respective vCPUs) that have to be placed to cover this demand, and, in turn, the price. In case of a surge

in traffic demand, no guarantees are provided and no service re-dimensioning takes place.

– *On-demand elastic service*: Following this model, the operator advertises guaranteed response time SLAs. The customer expresses an initial demand and the operator translates it to an initial number of VNF instances (and vCPUs). During the service lifetime, the number of ongoing HTTP connections per VNF instance is monitored via the service-level monitoring/management interface and, when necessary, the service provider decides to up- or down-scale the service in order to match the target response time distribution. For example, if the total number of parallel user sessions increases beyond a threshold such that the SLA objective cannot be met, a new instance is launched and user requests for content are appropriately redistributed.

Under the elastic model, the operator can apply a flexible on-demand pricing scheme, where a base price is derived from the initial CP service request, and the price is scaled following the amount of additional resources that need to be deployed to meet the target response times encoded in the agreed-upon SLA. Note that another pricing dimension has to do with video quality, which can be appropriately combined with the above pricing models. Finally, it should be pointed out that the reported response time distributions are mainly bounded by the CPU capabilities of the VNF platform and the host-to-guest communication overhead, and do not include the additional delay imposed in the user-VNF instance path. However, this path is monitored and controlled by the network operator, which can exploit internal information to derive a more accurate response time distribution (e.g., by shifting the curve appropriately to account for fixed network-level overheads), apply specific "safety margins" in the advertised response time guarantees, or accordingly provision this path with network resources to avoid SLA violations.

ACRONYMES

API Application Programming Interfaces.

BSS Business Support System.

CDN Content Delivery Network.

CDNaaS Content Delivery Network as a Service.

CP Content Provider.

ETSI European Telecommunications Standards Institute.

GA Genetic Algorithm.

GRNET Greek Research and Technology Network.

HTCP Hypertext Caching protocols.

ICP Internet Cache Protocol.

ILP Integer Linear Problem.

IM Instance Manager.

ISP Internet Service Provider.

MEC Mobile (or Multi-Access) Edge Computing.

MOS Mean Opinion Score.

NFV Network Functions Virtualisation.

NFVO NFV Orchestrator.

OSS Operation Support System.

PKI Public Key Infrastructures.

PMs Physical Machines.

PoP Points of Presence.

PSQA Pseudo-Subjective Quality Assessment.

QoE Quality of Experience.

QP Quantization Parameter.

REST Representational State Transfer.

RO Resource Orchestrator.

SDN Software Defined Networking.

SFC Service Function Chaining.

SIDR Service Instance Description Repository.

SIG Service Instance Graph.

SLA Service Level Agreement.

SLO Service Level Objective.

SMP Symmetric multiprocessing.

SO Service Orchestrator.

uRLLC ultra-Reliable Low-Latency Communication.

vCDN virtual Content Delivery Network.

VM Virtual Machine.

VNF Virtual Network Functions.

PUBLICATIONS FROM THE THESIS

International conferences

1. **L. Yala**, P. Frangoudis and A. Ksentini, "Latency and availability driven VNF placement in a MEC-NFV environment" *IEEE Globecom 2018. Accepted, to be published.*

2. **L. Yala**, P. Frangoudis, G. Lucarelli and A. Ksentini, "Balancing between cost and availability for CDNaaS resource placement" *IEEE Globecom 2017 .*

3. **L. Yala**, P. Frangoudis and A. Ksentini, "QoE-aware Computing Resource Allocation for CDN-as-a-Service Provision" *IEEE Globecom 2016.*

4. P. Frangoudis, **L. Yala**, A. Ksentini and T. Taleb, "An architecture for on-demand service deployment over a telco CDN" *IEEE ICC 2016.*

International Journals

1. P. A. Frangoudis, **L. Yala**, and A. Ksentini, "Cdn-as-a-service provision over a telecom operator's cloud" *IEEE Transactions on Network and Service Management*, 2017.

BIBLIOGRAPHY

[1] Akamai, *Aura licensed Akamai*, 2016b, URL: https://www.akamai.com/us/en/products/network-operator/licensed-cdn-solutions.jsp.

[2] Abdelhamid Alleg et al., "Delay-aware VNF Placement and Chaining based on a Flexible Resource Allocation Approach", *in*: *2017 13th International Conference on Network and Service Management (CNSM)*, IEEE, 2017, pp. 1–7.

[3] Fernando Almeida and Catalin Calistru, "The Role of Content Distribution Networks in the Future Media Networks", *in*: *International Journal of Emerging Trends and Technology in Computer Science* 1.*1* (2012), pp. 77–85.

[4] Faizul Bari et al., "Orchestrating virtualized network functions", *in*: *IEEE Transactions on Network and Service Management* 13.*4* (2016), pp. 725–739.

[5] Ilias Benkacem et al., "Optimal VNFs placement in CDN Slicing over Multi-Cloud Environment", *in*: *IEEE Journal on Selected Areas in Communications* (2018).

[6] *Big Buck Bunny*, URL: https://peach.blender.org/.

[7] Alain Billionnet and Frédéric Calmels, "Linear programming for the 0-1 quadratic knapsack problem", *in*: *European Journal of Operational Research* 92.*2* (1996), pp. 310–325.

[8] Norman Bobroff, Andrzej Kochut, and Kirk Beaty, "Dynamic placement of virtual machines for managing SLA violations", *in*: *Proc. IFIP/IEEE IM*, 2007.

[9] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads", *in*: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, IEEE, 2010, pp. 228–235.

[10] Mar Callau-Zori et al., *An experiment-driven energy consumption model for virtual machine management systems*, Research Report RR-8844, IRISA ; Université de Rennes 1 ; CNRS, Jan. 2016.

[11] Marco Casazza et al., "Securing virtual network function placement with high availability guarantees", *in*: *arXiv preprint arXiv :1701.07993* (2017).

[12] Lino Chamorro, Fabio López-Pires, and Benjamín Barán, "A genetic algorithm for dynamic cloud application brokerage", *in*: *Cloud Engineering (IC2E), 2016 IEEE International Conference on*, IEEE, 2016, pp. 131–134.

[13] Aleksandra Checko et al., “Cloud RAN for mobile networks-A technology overview”, *in*: *IEEE Communications surveys & tutorials* 17.*1* (2015), pp. 405–426.

[14] NM Mosharaf Kabir Chowdhury and Raouf Boutaba, “A survey of network virtualization”, *in*: *Computer Networks* 54.*5* (2010), pp. 862–876.

[15] VNI Cisco, *Forecast and Methodology, 2015–2020, white paper, Cisco, June 2016*.

[16] S. Clayman et al., “The dynamic placement of virtual network functions”, *in*: *Proc. IEEE NOMS*, 2014.

[17] *Content Delivery Network Interconnection (CDNI) Working group. IETF*. URL: https://datatracker.ietf.org/wg/cdni/documents/.

[18] Mouhamad Dieye et al., “CPVNF : Cost-efficient Proactive VNF Placement and Chaining for Value-Added Services in Content Delivery Networks”, *in*: *IEEE Transactions on Network and Service Management* 15.*2* (2018), pp. 774–786.

[19] *Docker*, URL: https://www.docker.com/.

[20] Jiankang Dong et al., “Energy-saving virtual machine placement in cloud data centers”, *in*: *Proc. IEEE/ACM CCGrid*, 2013.

[21] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni, “Ant system : optimization by a colony of cooperating agents”, *in*: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26.*1* (1996), pp. 29–41.

[22] Fred Douglis and M Frans Kaashoek, “Guest editors’ introduction : Scalable internet services”, *in*: *IEEE Internet Computing* 5.*4* (2001), p. 36.

[23] Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia, “Virtualization vs Containerization to Support PaaS”, *in*: *Proc. IEEE IC2E*, 2014.

[24] GSNFV ETSI and GSNFV ETSI, “V1. 1.1. Network Functions Virtualization”, *in*: *Terminology of Main Concepts in NFV* (2012).

[25] TS ETSI, *Telecommunications and IÂčnternet converged Services and Protocols for Advanced Networking (TISPAN) ; Content Delivery Network (CDN) Architecture*, Technical Specification, ETSI, 2011.

[26] Thibaud Ecarot, Djamal Zeghlache, and Cedric Brandily, “Consumer-and-Provider-Oriented Efficient IaaS Resource Allocation”, *in*: *Proc. 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017.

[27] *FP7 T-NOVA*, URL: http://www.t-nova.eu.

[28] Wes Felter et al., *An Updated Performance Comparison of Virtual Machines and Linux Containers*, tech. rep. RC25482, IBM Research, 2014, URL: http://goo.gl/ytEvt9.

[29] Md Hasanul Ferdaus et al., “An algorithm for network and data-aware placement of multi-tier applications in cloud data centers”, *in*: *Journal of Network and Computer Applications* 98 (2017), pp. 65–83.

[30] Pantelis A. Frangoudis, Louiza Yala, and Adlen Ksentini, “CDN-as-a-Service Provision over a Telecom Operator’s Cloud”, *in*: *IEEE Trans. Netw. Service Manag.* (2017), doi :10.1109/TNSM.2017.2710300.

[31] Pantelis A Frangoudis, Louiza Yala, and Adlen Ksentini, “CDN-as-a-Service Provision over a Telecom Operator’s Cloud”, *in*: *IEEE Transactions on Network and Service Management* 14.*3* (2017), pp. 702–716.

[32] Pantelis A. Frangoudis et al., “An architecture for on-demand service deployment over telco CDN”, *in*: *Proc. IEEE ICC*, 2016.

[33] Benjamin Frank et al., “Collaboration Opportunities for Content Delivery and Network Infrastructures”, *in*: *ACM SIGCOMM ebook on Recent Advances in Networking* 1 (2013).

[34] Benjamin Frank et al., “Pushing CDN-ISP collaboration to the limit”, *in*: *ACM SIGCOMM Computer Communication Review* 43.*3* (2013), pp. 34–44.

[35] G. Gallo, P. L. Hammer, and B. Simeone, “Quadratic knapsack problems”, *in*: *Combinatorial Optimization*, ed. by M. W. Padberg, Berlin, Heidelberg: Springer Berlin Heidelberg, 1980, pp. 132–149.

[36] Yongqiang Gao et al., “A multi-objective ant colony system algorithm for virtual machine placement in cloud computing”, *in*: *Journal of Computer and System Sciences* 79.*8* (2013), pp. 1230–1242.

[37] Daniel Gmach, Jerry Rolia, and Ludmila Cherkasova, “Resource and virtualization costs up in the cloud : Models and design choices”, *in*: *Proc. IEEE/IFIP DSN*, 2011.

[38] *Greek Research and Technology Network - Network Topology*, URL: https://mon.grnet.gr/pops/.

[39] Chuanxiong Guo et al., “Secondnet : a data center network virtualization architecture with bandwidth guarantees”, *in*: *Proceedings of the 6th International COnference*, ACM, 2010, p. 15.

[40] Nir Halman, Hans Kellerer, and Vitaly A. Strusevich, “Approximation Schemes for Non-Separable Non-Linear Boolean Programming Problems under Nested Knapsack Constraints”, *in*: *European Journal of Operational Research* (2018), in press, URL: https://www.sciencedirect.com/science/article/pii/S0377221718303163.

[41] Bo Han et al., “Network function virtualization : Challenges and opportunities for innovations”, *in*: *Communications Magazine, IEEE* 53.*2* (2015), pp. 90–97.

[42] Nicolas Herbaut, George Xilouris, and Daniel Négru, “The Surrogate vNF approach for Content Distribution”, *in*: *IEEE COMSOC MMTC E-Letter* 10.*4* (2015).

[43] Nicolas Herbaut et al., “Content Delivery Networks as a Virtual Network Function : A Win-Win ISP-CDN Collaboration”, *in*: *Proc. IEEE Globecom*, 2016.

[44] Zhe Huang and Danny HK Tsang, “SLA guaranteed virtual machine consolidation for computing clouds”, *in*: *Communications (ICC), 2012 IEEE International Conference on*, IEEE, 2012, pp. 1314–1319.

[45] Hatem Ibn-Khedher et al., “OPAC : An optimal placement algorithm for virtual CDN”, *in*: *Computer Networks* 120 (2017), pp. 12–27.

[46] *Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1 : Media presentation description and segment formats*, Standard, May 2014.

[47] Assaf Israel and Danny Raz, “Cost aware fault recovery in clouds”, *in*: *Proc. IFIP/IEEE IM*, 2013.

[48] Namita R Jain and Rakesh Rajani, “Virtualization Technology to Allocate Data Centre Resources Dynamically Based on Application Demands in Cloud Computing”, *in*: *IJSHRE* (2014).

[49] R. Jain and S. Paul, “Network virtualization and software defined networking for cloud computing : a survey”, *in*: *IEEE Communications Magazine* 51.*11* (2013), pp. 24–31.

[50] Fatma Ben Jemaa, Guy Pujolle, and Michel Pariente, “QoS-aware VNF placement optimization in edge-central carrier cloud architecture”, *in*: *Global Communications Conference (GLOBECOM), 2016 IEEE*, IEEE, 2016, pp. 1–7.

[51] Noriaki Kamiyama et al., “ISP-operated CDN”, *in*: *Proc. IEEE INFOCOM Workshops*, Rio de Janeiro, Brazil, 2009.

[52] Kostas Katsalis et al., “SLA-driven VM Scheduling in Mobile Edge Computing”, *in*: *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, IEEE, 2016, pp. 750–757.

[53] A. Kivity et al., “kvm : the Linux virtual machine monitor”, *in*: *Proc. Linux Symposium*, 2007.

[54] Adlen Ksentini et al., “Providing low latency guarantees for slicing-ready 5G systems via two-level MAC scheduling”, *in*: *IEEE Network* (2018), in press.

[55] Stefan Lederer, Christopher Müller, and Christian Timmerer, “Dynamic adaptive streaming over HTTP dataset”, *in*: *Proc. 3rd ACM MMSys*, 2012, pp. 89–94.

[56] Hyojung Lee, Dongmyung Lee, and Yung Yi, “On the economic impact of Telco CDNs and their alliance on the CDN market”, *in*: *Proc. IEEE ICC*, 2014.

[57] Harry R Lewis, "ΠGarey Michael R. and Johnson David S.. Computers and intractability. A guide to the theory of NP-completeness. WH Freeman and Company, San Francisco1979, x+ 338 pp.", *in*: *The Journal of Symbolic Logic* 48.*2* (1983), pp. 498–500.

[58] Xin Li and Chen Qian, "A survey of network function placement", *in*: *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*, IEEE, 2016, pp. 948–953.

[59] Xin Li and Chen Qian, "The virtual network function placement problem", *in*: *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*, IEEE, 2015, pp. 69–70.

[60] Zhe Li and Gwendal Simon, "In a Telco-CDN, pushing content makes sense", *in*: *IEEE Transactions on Network and Service Management* 10.*3* (2013), pp. 300–311.

[61] Ben Liang, *Mobile edge computing*, Cambridge University Press, 2017.

[62] Jenn-Wei Lin and Chien-Hung Chen, "Interference-aware virtual machine placement in cloud computing systems", *in*: *Computer & Information Science (ICCIS), 2012 International Conference on*, vol. 2, IEEE, 2012, pp. 598–603.

[63] Richard TB Ma, John CS Lui, and Vishal Misra, "Evolution of the internet economic ecosystem", *in*: *IEEE/ACM Transactions on Networking* 23.*1* (2015), pp. 85–98.

[64] Xiao Ma et al., "Cost-efficient workload scheduling in cloud assisted mobile edge computing", *in*: *Quality of Service (IWQoS), 2017 IEEE/ACM 25th International Symposium on*, IEEE, 2017, pp. 1–10.

[65] Patrick Maillé, Gwendal Simon, and Bruno Tuffin, "Vertical integration of CDN and network operator : Model and analysis", *in*: *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, IEEE, 2016, pp. 189–195.

[66] Zoltán Ádám Mann, "Allocation of Virtual Machines in Cloud Data Centers–A Survey of Problem Models and Optimization Algorithms", *in*: *ACM Comput. Surv.* 48.*1* (2015), p. 11.

[67] R. Timothy Marler and Jasbir S. Arora, "Function-transformation methods for multi-objective optimization", *in*: *Engineering Optimization* 37.*6* (2005), pp. 551–570.

[68] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement", *in*: *INFOCOM, 2010 Proceedings IEEE*, IEEE, 2010, pp. 1–9.

[69] Rashid Mijumbi et al., “Design and evaluation of algorithms for mapping and scheduling of virtual network functions”, *in*: *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, IEEE, 2015, pp. 1–9.

[70] Rashid Mijumbi et al., “Network function virtualization : State-of-the-art and research challenges”, *in*: *IEEE Communications Surveys & Tutorials* 18.*1* (2016), pp. 236–262.

[71] Melanie Mitchell, *An Introduction to Genetic Algorithms*, 5th ed., Cambridge, MA, USA: MIT Press, 1999, ISBN: 0-262-63185-7.

[72] *Mobile Edge Computing (MEC) ; Deployment of Mobile Edge Computing in an NVF environment*, Group Report, 2018.

[73] *Mobile Edge Computing (MEC) ; Framework and Reference Architecture*, Group Specification, 2016.

[74] *Mobile Edge Computing (MEC) ; Mobile Edge Management ; Part 2 : Application lifecycle, rules and requirements management*, Group Specification, 2017.

[75] H. Moens and F. De Turck, “VNF-P : A model for efficient placement of virtualized network functions”, *in*: *Proc. IFIP CNSM*, 2014, pp. 418–423.

[76] *Network Functions Virtualisation (NFV) ; Management and Orchestration ; Report on Architectural Options*, Group Specification, Dec. 2014.

[77] *Network Functions Virtualisation (NFV) ; Management and Orchestration*, Group Specification, Dec. 2014.

[78] *Network Functions Virtualisation (NFV) ; Resiliency Requirements*, Group Specification, Jan. 2015.

[79] *Network Functions Virtualisation (NFV) ; Use Cases*, Group Specification, Oct. 2013.

[80] *Network Functions Virtualisation (NFV)-architectural framework*, Group Specification, Oct. 2013.

[81] *Nmap - the network mapper*, URL: https://nmap.org/.

[82] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun, “The akamai network : a platform for high-performance internet applications”, *in*: *ACM SIGOPS Operating Systems Review* 44.*3* (2010), pp. 2–19.

[83] *OpenStack - Open Source Cloud Computing Software*, URL: https://www.openstack.org/.

[84] Hamza Ouarnoughi et al., “A cost model for virtual machine storage in cloud IaaS context”, *in*: *Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro International Conference on*, IEEE, 2016, pp. 664–671.

[85] George Pallis and Athena Vakali, “Insight and perspectives for content delivery networks”, *in*: *Communications of the ACM* 49.*1* (2006), pp. 101–106.

[86] Al-Mukaddim Khan Pathan and Rajkumar Buyya, “A taxonomy and survey of content delivery networks”, *in*: ().

[87] Larry Peterson, Bruce Davie, and Ray van Brandenburg, *Framework for content distribution network interconnection (CDNI)*, tech. rep., 2014.

[88] Fabio López Pires and Benjamin Baran, “A virtual machine placement taxonomy”, *in*: *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, IEEE, 2015, pp. 159–168.

[89] Ingmar Poese et al., “Improving content delivery with PaDIS”, *in*: *IEEE Internet Computing* 16.*3* (2012), pp. 46–52.

[90] Long Qu et al., “Reliability-aware service provisioning in NFV-enabled enterprise datacenter networks”, *in*: *Proc. 12th International Conference on Network and Service Management (CNSM)*, 2016.

[91] Nguyen Quang-Hung, Nguyen Thanh Son, and Nam Thoai, “Energy-Saving Virtual Machine Scheduling in Cloud Computing with Fixed Interval Constraints”, *in*: *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXI*, Springer, 2017, pp. 124–145.

[92] W. Rankothge et al., “Optimizing Resource Allocation for Virtualized Network Functions in a Cloud Center Using Genetic Algorithms”, *in*: *IEEE Transactions on Network and Service Management* 14.*2* (2017), pp. 343–356.

[93] Gerardo Rubino, “Quantifying the Quality of Audio and Video Transmissions over the Internet : The PSQA Approach”, *in*: *Communication Networks & Computer Systems*, ed. by Javier A. Barria, Imperial College Press, 2005.

[94] Huzur Saran and Vijay V Vazirani, “Finding k cuts within twice the optimal”, *in*: *SIAM Journal on Computing* 24.*1* (1995), pp. 101–108.

[95] Vincenzo Sciancalepore et al., “A double-tier MEC-NFV Architecture : Design and Optimisation”, *in*: *Standards for Communications and Networking (CSCN), 2016 IEEE Conference on*, IEEE, 2016, pp. 1–6.

[96] *Series H : Audiovisual and Multimedia Systems – Infrastructure of audiovisual services – Coding of moving video – Advanced video coding for generic audiovisual services*, Recommendation, Feb. 2016.

[97] Kamal Deep Singh, Yassine Hadjadj Aoul, and Gerardo Rubino, “Quality of experience estimation for adaptive HTTP/TCP video streaming using H.264/AVC”, *in*: *Proc. IEEE CCNC*, 2012.

[98] Weijia Song et al., “Adaptive resource provisioning for the cloud using online bin packing”, *in*: *IEEE Trans. Comput.* 63.*11* (2014), pp. 2647–2660.

[99] S. Spagna et al., “Design principles of an operator-owned highly distributed content delivery network”, *in*: *IEEE Communications Magazine* 51.*4* (2013), pp. 132–140.

[100] T. Taleb et al., “EASE : EPC as a service to ease mobile core network deployment over cloud”, *in*: *IEEE Network* 29.*2* (2015), pp. 78–88.

[101] Tarik Taleb et al., “On multi-access edge computing : A survey of the emerging 5G network edge cloud architecture and orchestration”, *in*: *IEEE Communications Surveys & Tutorials* 19.*3* (2017), pp. 1657–1681.

[102] Shiqiang Wang et al., “Dynamic service migration in mobile edge-clouds”, *in*: *IFIP Networking Conference (IFIP Networking), 2015*, IEEE, 2015, pp. 1–9.

[103] Zhenyu Wen et al., “Cost effective, reliable and secure workflow deployment over federated clouds”, *in*: *IEEE Transactions on Services Computing* 10.*6* (2017), pp. 929–941.

[104] T. Wood et al., “Toward a software-based network : integrating software defined networking and network function virtualization”, *in*: *IEEE Network* 29.*3* (2015), pp. 36–41.

[105] G. Xilouris et al., “T-NOVA : A marketplace for virtualized network functions”, *in*: *Proc. EuCNC*, 2014.

[106] Jing Xu and Jose AB Fortes, “Multi-objective virtual machine placement in virtualized data center environments”, *in*: *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, IEEE Computer Society, 2010, pp. 179–188.

[107] Louiza Yala, Pantelis A. Frangoudis, and Adlen Ksentini, “QoE-Aware Computing Resource Allocation for CDN-as-a-Service Provision”, *in*: *Proc. IEEE Globecom*, 2016.

[108] Louiza Yala et al., “Balancing between cost and availability for CDNaaS resource placement”, *in*: *IEEE Global Communications Conference (GLOBECOM 2017)*, 2017.

[109] Song Yang, Philipp Wieder, and Ramin Yahyapour, “Reliable Virtual Machine placement in distributed clouds”, *in*: *Proc. 8th Int'l Workshop on Resilient Networks Design and Modeling (RNDM)*, 2016.

[110] Faqir Zarrar Yousaf et al., “SoftEPC - Dynamic instantiation of mobile core network entities for efficient resource utilization”, *in*: *Proc. IEEE ICC*, 2013.

[111] *ab - Apache HTTP server benchmarking tool*, URL: https://httpd.apache.org/docs/2.2/programs/ab.html.

[112] *weighttp*, URL: http://redmine.lighttpd.net/projects/weighttp/wiki.

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Les réseaux de diffusion de contenu

Mot clés : CDN, CDNaaS, vCDN, NFV, placement VNF/VM

Resumé : Le but de cette thèse est d'étudier et d'évaluer le role de la virtualisation des réseau de diffusion de contenu. Nous proposons une implementation d'une architecture CDN permettant à un opérateur de réseau de virtualiser son infrastructure CDN et de la louer à des fournisseurs de contenu. Afin d'avoir une allocation optimale des ressources, nous proposons une méthode qui combine les informations fournies lors de la demande par le fournisseur de contenu avec les données du réseau et de l'infrastructure de calcul. Nous avons modélisé ce problème d'allocation de ressources en problème d'optimisation, résolu par un algorithme. Les résultats obtenues donnent suite à la proposition d'algorithms et d'heuristiques de placement pour l'allocation conjointe de vCPU-à-VM et le placement des VMs dans les PMs.

Title : Content Delivery Network-as-a-Service (CDNaaS)

Keywords : CDN, CDNaaS, vCDN, NFV, VM/VNF placement

Abstract : The goal of this thesis is to study and evaluate the role a Virtual CDNs in improving the end-users QoE while saving on service providers' costs and service availability. First, we present the design and implementation of an architecture for on-demand deployment of a vCDN infrastructure over a telco cloud. Second, we propose different algorithms for solving the Virtual Network Function (VNF) placement problem. We propose a polynomial-time heuristic algorithms to solve a relaxed version of the problem's assumptions, we show experimentally that the derived solutions are close to the optimal. Finally, we study and evaluate solutions for the placement of VNF at the edge, by moving from the traditional central cloud to the edge one. We have also shown how our method can reduce delays and still provide a highly-available service.