Placement de services virtualisés par des stratégies intelligentes pour l'algorithme de Branch and Bound

Masoud Taghavian, Yassine Hadjadj-Aoul, Géraldine Texier, Philippe Bertin

Placement de services virtualisés par des stratégies intelligentes pour l'algorithme de Branch and Bound

Masoud Taghavian[12] et Yassine Hadjadj-Aoul[13] et Géraldine Texier [12] et Philippe Bertin[14]

[1] *IRT-BCOM, prénom.nom@b-com.com ;* [2] *IMT Atlantique/IRISA, prénom.nom@imt-atlantique.fr ;* [3] *University of Rennes, Inria, CNRS, IRISA, prénom.nom@irisa.fr ;* [4] *Orange, prénom.nom@orange.com*

La virtualisation des fonctions réseau (Network Function Virtualization, NFV) constitue un changement majeur, permettant de passer d'équipements matériels dédiés à des logiciels réutilisables fonctionnant dans des environnements virtualisés légers. Ces fonctions peuvent être déployées à la volée et nécessitent des stratégies de placement efficaces afin d'optimiser la gestion des ressources physiques partagées. Nous proposons une solution pour le placement systématique des services de réseau virtualisés, à la fois adaptative et proche des résultats optimaux, adaptée aux scénarios en ligne avec des contraintes de temps strictes. Notre solution, basée sur une recherche Branch and Bound, tire parti de stratégies de recherche de l'intelligence artificielle (en particulier A*) pour résoudre le problème du placement en maximisant l'acceptation de services (SA). L'analyse empirique approfondie réalisée montre la pertinence de cette approche.

Mots-clefs : Network function virtualization, Branch and Bound, A-Star

1 Introduction

L'interconnexion de fonctions réseaux virtualisées (VNFs) permet de composer des services réseaux complexes. Le problème du placement de graphe de VNFs (VNF-P) vise à trouver une implantation de la chaîne de services sur le réseau sous-jacent (SN). Cela nécessite des algorithmes sophistiqués capables d'allouer les ressources physiques de manière optimale, dans une contrainte de temps raisonnable. Ce problème de placement est NP-difficile, il s'agit en effet d'une généralisation du problème d'incorporation de réseau virtuel (Virtual Network Embedding (VNE)) ayant lui-même déjà été prouvé NP-difficile [3]. Une partie considérable de la littérature sur le problème de placement est consacrée à la résolution exacte du problème, généralement par programmation linéaire [1]. Bien que puissantes et optimales, ces techniques souffrent d'une complexité temporelle exponentielle et passent difficilement à l'échelle. Les méta-heuristiques [4] sont plus efficace, mais posent des problèmes de convergence et d'imprévisibilité. De plus, elles impliquent une recherche approfondie sur l'ensemble de l'espace des solutions. Or le problème de placement de graphes de VNFs est soumis à un grand nombre de contraintes (entre autres liées aux capacités des nœuds et des liens ou au respect de contraintes de qualité de service (QoS)), rendant une grande partie des solutions infaisables. L'utilisation de techniques d'Intelligence Artificielle (IA) et d'apprentissage pour le problème de placement est assez récente, mais très prometteuse [2].

Nous proposons une solution rapide et fiable de placement de services réseau virtualisés basée sur une formulation Branch and Bound (BB) s'appuyant sur différentes stratégies de recherche (notamment A*). Cela permet un compromis entre la recherche d'un placement proche de l'optimal et la rapidité de calcul. Notre contribution est adaptée aux scénarios de placement de services en ligne imposant une contrainte de temps. Une analyse empirique a été réalisée pour garantir l'efficacité de la solution proposée en évaluant l'acceptation de service (SA) pour comparer les stratégies en fonction du nombre de services placés avec succès.

2 Placement de graphes de services avec la méthode Branch-and-Bound

Le SN est représenté par un graphe dirigé de nœuds et de liens. Chaque nœud dispose d'une quantité disponible de ressources en CPU et en stockage. De même, nous considérons les ressources des liens en bande passante et en latence. Un service est représenté par un graphe dirigé (SG) composé de VNFs connectées par des liens virtuels (VL). Chaque VNF est un logiciel exécutable, qui nécessite un sous-ensemble des ressources disponibles d'un nœud du SN. Un VL est placé sur un chemin (une séquence de liens sans boucles) utilisant une partie des ressources disponibles de liens du SN. Un SG est associé à un contrat de service (SLA), qui reflète ses exigences de QoS (par exemple le délai de bout en bout, la disponibilité). Le placement est terminé lorsque tous les VNFs et les VLs du graphe de service sont placés. Nous considérons le placement en ligne des services, ce qui implique d'effectuer le placement dans un temps limité, proche du temps réel. La formalisation du problème de placement est présentée dans [6], la version étendue de ce papier.

L'algorithme BB est utilisé pour résoudre des problèmes d'optimisation combinatoire pouvant nécessiter l'exploration de toutes les permutations possibles dans le pire des cas. BB divise l'ensemble total des solutions réalisables en sous-ensembles plus petits et évite systématiquement les branches non prometteuses de l'arbre par élagage. L'approche BB que nous proposons est organisée comme une recherche arborescente. Les éléments d'un arbre de recherche sont appelés états. Leur profondeur est déterminée par la longueur du chemin entre la racine de l'arbre et l'état. Chaque état comprend un statut du SN (ressources restantes de chaque nœud et lien). Il hérite de la liste des VNFs et VLs non placés de son état parent, et porte un placement partiel pour un sous-ensemble de VNFs et de VLs du SG. Un état terminal est atteint lorsque le placement est terminé (tous les VNFs et VLs du SG sont placés avec succès sans violer de contraintes). La stratégie de recherche définit l'ordre de visite des états de l'arbre. Le parcours est effectué grâce à une liste ordonnée d'états, appelée frange, mise en œuvre par une file FIFO (pour Breadth-First Search (BFS)), une pile LIFO (pour Depth-First Search (DFS)) ou une file prioritaire (pour A*).

Au début, nous créons une liste triée de VNFs à partir d'un parcours BFS du SG demandé, en commençant par le point d'entrée (BFS garantit que nos placements partiels sont des sous-graphes connectés du SG). Nous instancions l'état de la racine de l'arbre et le plaçons dans la frange avant d'entrer dans une boucle de recherche. À chaque itération, nous sortons un état de la frange et vérifions s'il s'agit d'un état terminal. Sinon, l'état est étendu plaçant une nouvelle VNF de la liste sur chacun des nœuds du SN. Au fur et à mesure du placement des VNFs, nous pouvons placer de nouveaux VLs dans le SN (un VL peut être placé si ses VNFs source et destination sont déjà placées). L'expansion donne une liste d'états enfants, qui sont des placements candidats d'une VNF spécifique sur différents noeuds SN. Les enfants qui ne violent aucune contrainte sont ajoutés à la frange pour être développés plus tard. Les autres sont élagués. La recherche continue jusqu'à trouver un état terminal (succès) ou si le temps de recherche accordé est écoulé (échec). La profondeur de l'arbre est égale au nombre de VNFs du SG, et le facteur de branchement est égal au nombre de noeuds du SN. L'objectif SA permet de choisir parmi les placements possibles. Il est défini comme le nombre de services qui peuvent être placés avec succès sur le SN.

2.1 Stratégies d'exploration de l'arbre de recherche

Nous comparons cinq stratégies de parcours de l'arbre de recherche afin d'étudier leur influence respective sur la qualité de la solution : A^* Bandwidth Optimized (*ABO*), DFS Bandwidth Optimized (*DBO*), A^* and DFS combined Bandwidth Optimized (ADBO), Enhanced Decreasing First-Fit (EDFF) and Enhanced Increasing First-Fit (EIFF). Alors que EDFF et EIFF se concentrent sur les ressources des nœuds, ABO, DBO et ADBO se concentrent sur les ressources des liens (la bande passante). La somme des ressources requises dans les nœuds par les VNFs du SG étant identique pour les cinq stratégies, nous nous concentrons sur les ressources des liens.

A* est un algorithme de recherche souvent utilisé en informatique en raison de sa complétude, de son optimalité et de son efficacité. Il traverse l'arbre de recherche en fonction de f-coûts qui sont calculés par $f(n) = g(n) + h(n)$ [5]. f, l'estimation du coût d'une solution de la racine à un état

cible, est la somme d'une fonction g donnant le coût réel du chemin de la racine à l'état actuel, et d'une fonction heuristique h estimant le coût de l'état actuel à l'état cible. La complexité spatiale exponentielle de A^* n'est pas un problème ici, la durée de la recherche étant bornée.

A* Bandwidth Optimized (ABO) utilise A^* et définit le coût comme la quantité de bande passante utilisée par le placement. La frange est rangée dans une file d'attente prioritaire. Ses états sont triés en fonction des f-coûts avec l'état dont le f-coût est le plus faible en tête. Le coût du placement peut être estimé par une heuristique optimiste en plaçant chaque VL sur un seul lien du SN et pas sur un chemin. Par conséquent, h est la quantité totale de bande passante requise par tous les VLs non placés, et g est la quantité totale de bande passante utilisée pour tous les VLs placés.

DFS Bandwidth Optimized (*DBO*) définit le coût comme la quantité de bande passante utilisée par le placement. L'arbre de recherche est traversé profondeur (DFS). Après avoir étendu un état, la liste des états générés est triée en fonction de leur coût et est placée dans la frange, ici une pile. Cela assure de développer en premier l'état ayant la plus faible utilisation de bande passante. *DBO* ne garantit pas nécessairement l'optimalité ou la complétude, mais il trouve un placement optimisé en termes de bande passante plus rapidement que *ABO*.

A* and DFS combined Bandwidth Optimized (ADBO) cherche un placement optimal avec ABO dans un temps borné pour permettre un placement en ligne, puis utilise DBO en cas de recherche infructueuse afin de trouver un placement quasi optimal.

Enhanced Decreasing First-Fit (EDFF) and Enhanced Increasing First-Fit (EIFF) améliorent deux heuristiques bien connues de placement de services qui trient les nœuds et placent la VNF sur le nœud ayant respectivement la plus grande (pour Decreasing First-Fit (DFF) (*aka* Best-Fit)) ou la plus petite (pour Increasing First-Fit (IFF)) quantité de ressources disponibles (mais suffisante). EDFF (respectivement EIFF) est mis en œuvre par un tri ascendant (respectivement descendant) des états générés en fonction de leur quantité de ressources disponibles (au lieu de leur utilisation de la bande passante). Les évaluations montrent que EDFF et EIFF peuvent placer en moyenne près de 3 fois plus de SGs que DFF et IFF.

3 Experimentations

Afin qu'une défaillance de nœud ne compromette pas la disponibilité des services, nous respectons la contrainte de ne pas placer deux VNFs d'un SG sur un seul nœud du SN dans nos évaluations. Une étude de l'impact de cette contrainte est disponible dans [6].

Les violations des métriques de QoS étant systématiquement détectées et prises en compte dans le BB, nous évaluons l'impact des stratégies de recherche en nous concentrant sur les ressources. Pour placer un VL sur un chemin du SN, nous utilisons un routage selon le chemin de plus petite longueur calculé avec l'algorithme de Dijkstra. Nos évaluations sont effectuées en initialisant les nœuds et les liens du SN avec le maximum de ressources disponibles avant d'entrer dans une boucle. À chaque itération, nous créons un SG selon la topologie, la taille et les ressources requises par les VNFs et les VLs. Ensuite, nous essayons de placer ce graphe de service Graph (SG) en utilisant une stratégie. Nous limitons les recherches de placements à une durée de 2000 ms. Si la stratégie a trouvé un placement possible, alors il est appliqué au SN en mettant à jour les ressources restantes puis nous commençons une nouvelle itération. En cas d'échec, l'évaluation se termine et renvoie le nombre de SG qui ont pu être placés.

Nos évaluations, réalisées en Java, sont exécutées dans un seul thread sur une machine sous Windows 10 dotée d'un processeur Intel Core i7-3687 et de 8 Go de RAM. Nous effectuons nos évaluations sur les topologies de SN BT-Europe, BT-Asie-Pacifique et BT-Amérique du Nord disponibles dans Zoo Topology. Elles comportent respectivement 24, 20 et 36 nœuds et 74, 62 et 152 liens bidirectionnels. Chaque nœud a des ressources illimitées de CPU et de stockage. Chaque lien dispose initialement de 10 unités de bande passante. Les SG sont composés de 3 à 8 VNFs utilisant chacune une unité de CPU et une unité de stockage, et de VLs bidirectionnels nécessitant 1 à 10 unités de bande passante. Nous évaluons des topologies en chaîne, en anneau ou en étoile.

Table 1: Amélioration du SA avec ADBO par rapport aux autres stratégies (%)

	Q1	Q2	Q3	Min	Moy.	Max
ABO	0	0	0	0	**7**	110
DBO	0	8	33	-33	**128**	3400
EDFF	60	100	125	-28	**184**	2300
EIFF	67	92	125	0	**211**	3400
DFF	200	300	500	60	**429**	2300
IFF	300	500	700	100	**564**	2200

Table 2: Comparaison de ABO, DBO et ADBO

taille	Nombre de SG placés			% bande passante restante		
SG	ABO	DBO	ADBO	ABO	DBO	ADBO
3	180	173	180	2.70	2.16	2.70
4	115	101	115	5.41	9.46	5.41
5	78	78	78	13.51	8.38	13.51
6	**60**	**57**	60	**18.92**	**12.97**	18.92
7	55	46	55	6.76	12.16	6.76
8	**30**	**38**	39	**43.24**	**7.03**	17.57

Le tableau 1 compare les stratégies en prenant ADBO comme référence. On observe que ADBO donne presque toujours de meilleurs résultats. Nous utilisons les quartiles (Q1, Q2 (médiane), Q3), le minimum, la moyenne et le maximum pour représenter les résultats. Une amélioration de 400% signifie que ADBO peut placer 5 fois plus de SGs que la stratégie évaluée. Au-delà de ces résultats satisfaisants en termes de moyenne, nous pouvons constater que pour les valeurs maximums, ADBO a pu placer 24 à 35 fois plus de SGs. Le problème de placement est alors si compliqué que DBO, EDFF et EIFF n'ont pu en trouver au mieux que 1 ou 2. Les placements réussis sont exécutés en un temps relativement court. Bien que cela dépende de la taille des SG et SN, en moyenne, ADBO, ABO, DBO, EDFF et EIFF peuvent placer un service en 53, 52, 40, 43 et 46 ms respectivement.

Le tableau 2 montre le nombre de SG placés ainsi que le pourcentage de la bande passante totale restante sur le SN pour les stratégies ABO, DBO et ADBO. Nous plaçons des SG de différentes tailles ayant une topologie en chaîne sur le réseau BT-Europe. Pour des SGs de 6 VNFs, ABO peut placer plus de SGs que DBO (60 contre 57), tout en laissant plus de bande passante disponible dans le SN (18,92% contre 12,97%). Cependant, pour des SGs de 8 VNFs, bien que ABO puisse laisser une grande quantité de bande passante disponible (43,24% contre 7,03%), il place moins de SGs (30 contre 38) en raison du temps de calcul borné. L'ensemble des résultats des évaluations montre l'intérêt de combiner ABO et DBO en une stratégie ADBO partageant le délai de calcul autorisé pour exécuter ABO puis DBO.

4 Conclusion

Le problème du placement de VNF existe depuis plusieurs années, or la recherche d'une solution de placement en ligne nécessite un compromis entre le temps de calcul, l'optimalité du placement et son évolutivité (son impact sur l'admission de services futurs). Nous avons proposé une solution basée sur l'algorithme Branche and Bound, tirant parti des stratégies de recherche de l'IA, en particulier A^*, en utilisant une heuristique admissible et cohérente, capable de trouver des placements optimaux, relativement rapidement. Une analyse empirique importante a été réalisée et les résultats confirment une amélioration considérable (429% en moyenne) par rapport à la méthode heuristique de placement Best-Fit. Par la suite, nous envisageons de poursuivre en étudiant les effets des métriques de qualité de service sur le problème du placement des VNF et l'exploration de réseaux SN de type Edge/Core, où typiquement nous manquons de ressources en périphérie.

Références

[1] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte. Orchestrating virtualized network functions. *IEEE TNSM*, 13(4) :725–739, 2016.

[2] D. M. Manias, H. Hawilo, M. Jammal, and A. Shami. Depth-optimized delay-aware tree (do-dat) for virtual network function placement. *IEEE Networking Letters*, 2(3) :149–153, 2020.

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network function virtualization : State-of-the-art and research challenges. *IEEE Communications surveys & tutorials*, 18(1) :236–262, 2015.

[4] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and S. Davy. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In *NetSoft 2015*, pages 1–9. IEEE, 2015.

[5] S. Russell and P. Norvig. *Artificial intelligence : a modern approach*. Prentice Hall, 2002.

[6] M. Taghavian, Y. Hadjadj-Aoul, G. Texier, and P. Bertin. An approach to network service placement using intelligent search strategies over branch-and-bound. In *GLOBECOM 2021*, pages 1–7, 2021.