UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ALLOCATION DES RESSOURCES DANS UNE INFRASTRUCTURE DE *FOG COMPUTING* DYNAMIQUE

THÈSE
PRÉSENTÉE
COMME EXIGENCE PARTIELLE
DU DOCTORAT EN INFORMATIQUE

PAR
AMINA MSEDDI

JANVIER 2026

REMERCIEMENTS

Il me sera très difficile de remercier tout le monde, car c'est grâce à l'aide de nombreuses personnes que j'ai pu mener cette thèse à son terme.

Mes premiers remerciements s'adressent à la Professeure Halima Elbiaze, ma directrice de recherche pour m'avoir encadrée et guidée tout au long de ma thèse ainsi que pour l'expérience enrichissante et pleine d'intérêt qu'elle m'a offerte. Je la remercie pour ses multiples conseils et pour toutes les heures qu'elle a consacrées à me diriger dans ma recherche. Je suis ravie d'avoir travaillé en sa compagnie, car outre son appui scientifique, elle a toujours été là pour me soutenir et me conseiller au cours de l'élaboration de cette thèse. Je la remercie également pour sa bonne humeur et sa convivialité, notamment pour ses qualités humaines d'écoute et de compréhension tout au long de ce travail, notamment lors des dures épreuves de la vie que j'ai vécues au cours de ce doctorat.

Je remercie aussi Docteur Wael Jaafar qui m'a énormément aidée dans la réalisation de cette thèse et pour ses remarques enrichissantes, et pour ses précieux conseils apportés lors des différentes étapes du doctorat.

Je tiens particulièrement à remercier Professeur Wessam Ajib pour ses nombreux conseils. Je le remercie aussi pour son esprit ouvert, son franc parler et la confiance qu'il a pu m'accorder durant ces années.

Mes plus sincères remerciements vont à mes parents, Hefedh et Saloua, pour leurs encouragements tout au long de mes études et leur soutien. Qu'ils trouvent, dans la réalisation de ce travail, l'aboutissement de leurs efforts ainsi que l'expression

de ma plus affectueuse gratitude. Je remercie en particulier mon mari Aymen, pour son soutien indéfectible et la confiance qu'il a toujours placée en moi. Je le remercie pour son aide, son écoute et surtout son amour qui m'a été essentiel. Je remercie ma sœur Asma, pour m'avoir fait partager sa joie de vivre et m'avoir ainsi soutenu dans mes efforts. Je ne pourrais oublier dans mes remerciements, mes deux petits enfants Lili et Jo , c'est grâce à leur amour inconditionnel et leurs sacrifices que j'ai pu terminer ce travail. Je voudrais aussi remercier ma belle-mère Rakia, ma tante Radhia, mon amie Khadija et mon beau-frère Mehdi pour leur encouragement continuel.

Bien sûr, je ne voudrais pas oublier dans mes remerciements mes collègues de l'UQAM : Mouhamad, Cirine, Yosr, Zakaria, Zoubeir, Amina, Khalil et Salah avec qui j'ai passé de très bons moments et qui m'ont surtout permis de travailler dans une ambiance de travail relaxante et propice à une bonne concentration.

Un remerciement spécial va aux membres de jury d'avoir accepté d'évaluer ma thèse.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ABRÉVIATIONS

CC Cloud Computing

COMA Conterfactual Multi-Agent

D2D Device To Device

DQN Deep Q-Network

DRL Deep Reinforcement Learning

DRQN Deep Recurrent Q-Network

FC Fog Computing

FCV Fog Computing Véhiculaire

FN Fog Node

IoT Internet of Things

IoT Internet of Things

IQL Independant Q-Learning

MARL Multi-Agent Reinforcement Learning

MDP Markov Decision Process

NF Nearest Fog

OEP Optimisation par Essaims Particulaires

OEP Optimisation à Essaims Particulaires

QoS Quality of Service

RF Random Fog

RL Reinforcement Learning

RSU RoadSide Unit

SLA Service Level Agreement

UE User Equipement

VDN Value Decomposition Networks

RÉSUMÉ

Le *fog computing* est une infrastructure distribuée qui permet d'étendre l'architecture du *cloud computing* vers un réseau de nœuds périphériques proches des utilisateurs. Cette infrastructure est composée d'un grand nombre d'appareils distribués et hétérogènes qui communiquent et coopèrent entre eux afin d'exécuter des services ou de stocker des données en contrepartie d'une récompense. Toutefois, la gestion de la mobilité, de l'état de la batterie et de la charge de calcul de ces nœuds est assurée par des entités distinctes. En effet, le comportement dynamique de ces nœuds physiques n'est pas contrôlé par le *fog*. Cependant, ces caractéristiques influent sur les processus d'allocation des ressources aux applications du *fog*. D'autre part, ces décisions doivent être prises en temps réel et tiennent en compte de la distribution des nœuds et de leur extensibilité.

L'objectif de cette thèse consiste à concevoir et à implémenter des stratégies d'allocation des ressources pour une infrastructure de *fog computing* dynamique. En outre, ces stratégies prennent en considération l'aspect dynamique, hétérogène, largement distribué et très extensible des nœuds physiques composant l'infrastructure du *fog*. Ces stratégies visent à maximiser le nombre de requêtes dont les délais de réponse sont inférieurs à certains seuils prédéfinis. Ces stratégies visent aussi à balancer la charge entre les différents nœuds. Pour ce faire, nous modélisons le problème d'allocation de ressources en plaçant les conteneurs exécutant les applications des utilisateurs dans les nœuds dynamiques du *fog*. Ensuite, nous proposons des stratégies centralisées et distribuées pour résoudre le problème sus-mentionné. Puis, nous évaluons les performances de ces stratégies et les avons comparées à d'autres approches de l'état de l'art. Les résultats

obtenus illustrent la supériorité des approches proposées par rapport aux méthodes conventionnelles en termes de taux de réussite.

Mots clés : Fog computing, allocation des ressources, placement des conteneurs, délai de réponse, NP-Difficulté, programmation linéaire en nombres entiers, algorithmes gloutons, optimisation par essaims particulaires, algorithmes d'apprentissage

CHAPITRE I

INTRODUCTION

Ce chapitre introduit le projet de ma thèse de doctorat. Il est divisé en six parties : (i) les motivations et le cadre du travail (ii) les problématiques (iii) les objectifs (iv) la méthodologie (v) les contributions, et finalement (vi) l'organisation de la thèse.

1.1 Motivations et cadre de la thèse

1.1.1 Les limitations du *cloud* face aux applications IoT

L'émergence de l'Internet des Objets (IoT pour "Internet of Things" en anglais) visant à relier de multiples types d'appareils à Internet permet de mettre en place diverses applications, telles que le transport intelligent, la réalité augmentée et la technologie vestimentaire. Cependant, ces appareils se caractérisent par des capacités de calcul et d'énergie limitées (Jiang *et al.*, 2017). À cet effet, une partie du traitement des applications s'exécutant sur ces appareils peut être déplacée vers les puissants serveurs des clouds (Liu *et al.*, 2013).Selon une nouvelle étude d'IHS Markit, le nombre d'appareils connectés atteindra 127 milliards d'ici 2030 (News, 2017). En outre, ces appareils généreront une quantité massive de données qui sera envoyée vers les centres de données du *cloud* et surchargera les réseaux de ceux-ci (Mukherjee *et al.*, 2018). De plus, ces centres de données sont généralement

déployés d'une manière globalement distribuée et éloignée des utilisateurs des applications IoT. En effet, des études ont montré que les délais de communication moyens entre les appareils intelligents générant les données et le centre de données où le traitement est exécuté sont aux alentours de 100 ms (Satyanarayanan *et al.*, 2009). Ces délais ne répondent pas aux exigences des applications sensibles en termes de latence (Jiao *et al.*, 2013), telles que les applications de réalité augmentée, les véhicules intelligents, la détection d'incendie, et les applications vidéo nécessitant des analyses en ligne (Mouradian *et al.*, 2018). Finalement, pour des exigences de sécurité, certaines données peuvent ne pas pouvoir être traitées dans des régions géographiques différentes de leurs origines. Ainsi, le fait de s'appuyer uniquement sur le *cloud*, ne peut pas satisfaire toutes les exigences des applications IoT actuelles. Par conséquent, le concept de *fog computing* (FC) a été introduit, permettant d'exécuter certains segments d'applications dans des nœuds périphériques plus proches des appareils des utilisateurs de l'IoT (Vaquero et Rodero-Merino, 2014).

1.1.2 Qu'est ce que le *fog computing* ?

Le *fog computing* est un paradigme introduit par Cisco (Cisco, 2015), permettant d'étendre l'architecture du *cloud* vers la périphérie du réseau. Il bénéficie désormais du soutien du groupe Openfog (Group, 2017). Le *fog* est un système où un très grand nombre d'appareils distribués et hétérogènes communiquent et coopèrent potentiellement entre eux afin d'exécuter des services ou stocker des données. Ces services peuvent être des fonctions réseau virtualisées ou des applications IoT. Dans le *fog computing*, les utilisateurs peuvent offrir une partie des ressources de leurs appareils en contre-partie d'une récompense (Vaquero et Rodero-Merino, 2014). Ces appareils qu'on appellera par la suite les nœuds du *fog*, sont plus proches des utilisateurs finaux que les serveurs du *cloud* et permettent ainsi de mieux répondre aux exigences des applications sensibles à la latence

(Mahmud *et al.*, 2018). La Figure 1.1.2 décrit un système de *fog computing* composé de trois couches. La première couche est composée des appareils des utilisateurs finaux et IoT. La seconde couche est constituée des appareils du *fog computing*. Finalement, la couche supérieure est représentée par les centres de données du *cloud computing*. Ces centres représentent le cœur du système, où les données et l'état du système sont stockés et parfois gérés.

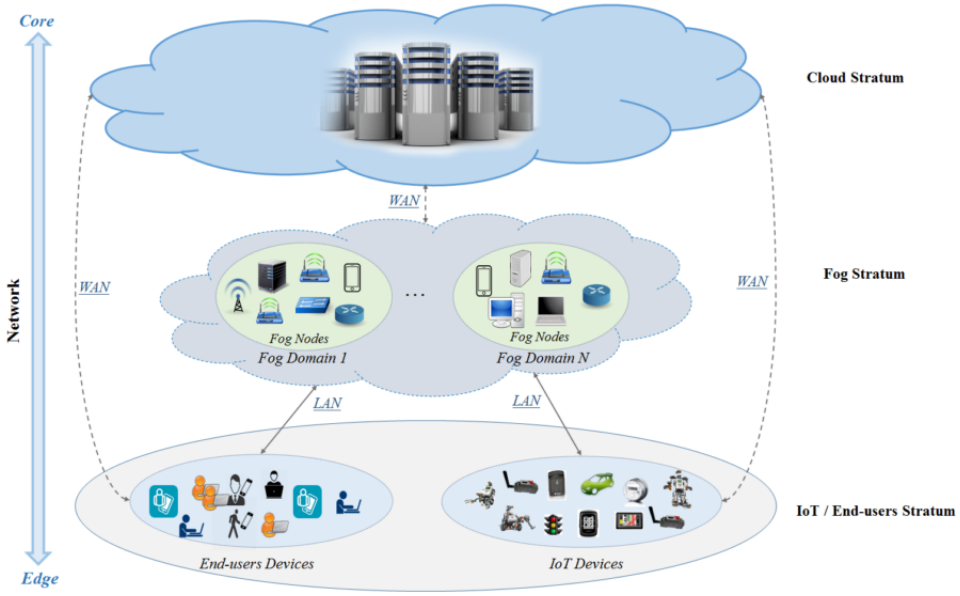


FIGURE 1.1 Le système du *fog computing* (Mouradian *et al.*, 2018)

1.1.3 Les caractéristiques d'un orchestrateur de ressources du *fog*

D'une part, le *fog computing*, se présentant comme une extension du *cloud* vers un réseau de nœuds périphériques proches des utilisateurs, hérite des caractéristiques majeures du *cloud*, telles que l'orchestration des ressources, l'extensibilité et la notion de multi-locataires (Group, 2017). D'autre part, le *fog computing*, requiert la prise en compte de la localisation des nœuds où les applications vont être déployées, de la mobilité des utilisateurs et des interactions en temps réel avec

les utilisateurs finaux. Il est important de clarifier que les ressources considérées peuvent référer à toute composante dans un nœud du *fog* pouvant être accessible et géré par l'orchestrateur à travers une interface. Ceci pourrait inclure la mémoire vive, la CPU, le réseau, les micro-services, les machines virtuelles, les conteneurs, etc. (De Brito *et al.*, 2017). En outre, une plateforme d'orchestration des ressources est la clé de voûte d'une architecture de *fog computing*. Cet orchestrateur permet principalement d'allouer des ressources de calcul et de communication dans les nœuds du *fog* dans le but d'y déployer les applications des utilisateurs finaux. L'orchestrateur doit aussi répondre aux exigences de qualité de service des applications en termes de délai, de débit et de localisation des données. À cet effet, il doit décider si les applications peuvent être déployées dans le *cloud* ou dans le *fog* et déterminer le nœud physique du *fog* où elles devraient être déployées. Pour ce faire, l'orchestrateur doit connaître en temps réel les disponibilités actuelles et futures des ressources des nœuds qu'il contrôle. Ceci requiert ainsi d'automatiser le processus de découverte des ressources. Ce processus étant en lien direct avec les nœuds du *fog*, il doit aussi tenir en compte de leurs caractéristiques telles que la mobilité, la variation de charge et l'état de la batterie. Finalement, l'architecture de cet orchestrateur doit prendre en compte la distribution géographique étendue et l'ajout et la disponibilité sporadique des nœuds du *fog* (Jiang *et al.*, 2017). L'exécution des tâches d'un orchestrateur est assurée par un ou un ensemble de nœuds du fog. Ces nœuds vont être appelés contrôleurs du fog. Leur topologie et leur interconnexion représenteront l'architecture de haut niveau de l'orchestrateur. De plus, le choix des nœuds physique de contrôle influence les délais de contrôles.

1.2 Problématique

D'une part, les nœuds du *fog* sont géographiquement distribués, mobiles et la disponibilité de leurs ressources est sporadique. D'autre part, les utilisateurs qui

communiquent avec cette infrastructure de *fog* sont aussi hétérogènes, et la charge dûe à leurs requêtes variable. Ces aspects dynamiques influencent les performances de l'infrastructure du fog computing en termes de délais de contrôle et de réponse aux requêtes des utilisateurs. Ainsi, les algorithmes d'allocation des ressources aux nœuds du *fog computing* devrait tenir en compte des caractéristiques de ces nœuds.

1.2.1 Problématique générale : les caractéristiques des nœuds du *fog computing*

Les nœuds du *fog computing* sont les dispositifs ou les infrastructures dotés de ressources de calculs, de stockage et de communication et se trouvant à la périphérie du réseau. En échange d'une compensation, ces nœuds sont capables de mettre une portion de leurs ressources à disposition pour des applications normalement exécutées sur les appareils des utilisateurs finaux ou dans le *cloud*. (Yi *et al.*, 2015).

Ces dispositifs peuvent appartenir à des entités distinctes. Leurs ressources peuvent être totalement ou partiellement dédiées à l'infrastructure du *fog*. Plusieurs types de nœuds hétérogènes ont été cités dans la littérature (Cisco, 2015; Marín-Tordera *et al.*, 2017) tels que les routeurs, commutateurs, points d'accès, caméras de vidéo-surveillance, serveurs, petits centres de données, téléphones intelligents, ordinateurs personnels, voitures, etc. Toutefois, ces nœuds ont un comportement dynamique et non contrôlé par le *fog*. En effet, seules les entités qui possèdent ces nœuds peuvent gérer la mobilité, l'état de la batterie et la charge de calcul de ces nœuds (Jamil *et al.*, 2022; Ningning *et al.*, 2016). Finalement, ces nœuds sont disséminés géographiquement et la disponibilité de leurs ressources est sporadique (Mukherjee *et al.*, 2018).

Les plateformes d'orchestration de ressources dédiées au *cloud* ne sont pas appropriées pour le *fog*. En effet, la nature dynamique et hétérogène des nœuds

du *fog* rend leur gestion différente, contrairement aux serveurs du *cloud* qui sont totalement dédiés à ce dernier. De plus, les serveurs sont immobiles, leurs ajouts et suppressions ne sont pas très fréquents et ils ont une quantité considérable de ressources. Par conséquent, les nouvelles plateformes d'orchestration dédiées au fog doivent tenir compte de la nature dynamique des nœuds.

1.2.2 Problématique spécifique : allocation des ressources dans le *fog*

Afin de répondre aux exigences en termes de latence, les applications sont exécutées dans les nœuds du *fog* au lieu d'être exécutées dans les serveurs du *cloud*. Un orchestrateur devra donc allouer les ressources convenables pour ces applications dans le but de satisfaire certains critères tels que l'accord de niveau de service (SLA pour Service Level Agreement en anglais). Plusieurs travaux ont proposé des plateformes et des algorithmes permettant d'allouer les ressources du *fog* aux différentes applications. Cependant, à notre connaissance, aucun de ces travaux ne considère la nature très dynamique des nœuds du *fog*. Or, une allocation qui ne prend pas en compte ce dynamisme influencerait d'une manière négative les performances des applications. Par exemple, un nœud physique du *fog* qui s'éloigne de l'utilisateur final de l'application demandée engendre une augmentation des délais de communications. De plus, lors de l'éloignement, l'extinction ou la surcharge d'un nœud, les applications qui y étaient exécutées doivent être migrées ou redéployées dans de nouveaux nœuds. Cette migration ou ce nouveau déploiement augmente aussi les délais de réponse à la demande de l'utilisateur final. Ajoutons à cela que si les appareils représentant les nœuds ne sont pas totalement consacrés au système du *fog*, ils sont prioritairement dédiés à leurs fonctions de base, puis aux applications du *fog computing* (Varghese *et al.*, 2017). Ainsi, la variation de la charge de traitement de ces fonctions de base a une influence sur les ressources disponibles au système de fog. Une hausse soudaine de cette charge va impliquer la migration des applications vers d'autres nœuds,

ajoutant ainsi des délais supplémentaires. En effet, une distribution efficace de la charge des applications du *fog* est nécessaire pour éviter les migrations inutiles et minimiser les délais de réponse. Ainsi, la considération de l'aspect dynamique des nœuds est essentielle pour optimiser les performances des applications en termes de délais de réponses.

1.2.3 Problématique spécifique : architecture de haut niveau des nœuds de contrôle

Dans une plateforme d'orchestration des ressources pour une infrastructure de *fog computing*, un seul ou un ensemble de nœuds doivent assurer la gestion des ressources, tout en assurant aussi la communication avec les utilisateurs finaux et le *cloud*. Dans un langage plus approprié, ceux-ci pourraient être appelés les contrôleurs des nœuds du *fog*. Nous appellerons par la suite "domaine de contrôle", l'ensemble des nœuds physiques contrôlés par un même contrôleur. En outre, la délimitation des domaines de contrôle va influencer le processus d'allocation des ressources. La modélisation de contrôleurs centraux comme ceux des plateformes d'orchestration conventionnelles du *cloud* ne répondent pas aux exigences des systèmes de *fog computing*. D'abord, la distribution géographique des nœuds du *fog* requiert aussi une distribution des contrôleurs. De plus, l'ajout fréquent et rapide de nœuds requiert une extensibilité efficace des contrôleurs. En outre, ces contrôleurs devraient collaborer pour la gestion de leurs domaines respectifs et pour optimiser le traitement des requêtes des utilisateurs.

1.2.4 Problématique spécifique : le seuil du délai de réponse

Les applications sensibles en termes de latence telles que les applications de réalité augmentée, les véhicules intelligents, la détection d'incendie, et les applications vidéo nécessitant des analyses en ligne (Mouradian *et al.*, 2018), sont des applications typiques qui devraient être déployées dans les environnements

de *fog computing* pour bénéficier d'un traitement continu en temps réel. Pour ces applications, plusieurs facteurs tels que le délai de réponse (ou de service), l'énergie, le contexte utilisateur-application jouent un rôle important dans l'approvisionnement en ressources dans l'environnement de *fog computing* (Mahmud *et al.*, 2018). En outre, le délai de réponse est considéré comme l'un des facteurs importants pour mesurer la qualité de service des applications déployées dans le *fog* (Haghi Kashani *et al.*, 2020). Ce délai est généralement divisé en trois composantes.

- Le délai de calcul qui correspond au temps nécessaire à l'exécution d'une tâche. Il dépend étroitement de la configuration des ressources où l'application ou la tâche sont exécutées (Zeng *et al.*, 2016) et peut varier en fonction de la charge existante du nœud (Jalali *et al.*, 2016).
- Le délai de communication ; qui définit essentiellement le délai requis pour échanger des éléments de données entre les utilisateurs finaux et les nœuds du *fog*. Le temps de communication requis est reflété par le contexte du réseau, ce qui facilite la sélection des nœuds du *fog* appropriés pour l'exécution des tâches.
- Le temps de gestion et d'orchestration. Ce délai englobe les processus de prise de décision concernant où, quand et comment les tâches doivent être exécutées dans l'infrastructure du fog. Cela inclut la sélection des nœuds du fog appropriés, la répartition des charges de travail, et la coordination des ressources entre les différents niveaux de l'infrastructure, de l'edge au cloud central. (Rimal *et al.*, 2018). Ce temps pourrait avoir un impact sur le délai de réponse.

Le seuil du délai de réponse spécifie la valeur maximale pour la livraison d'un service ou de la réponse à une requête qu'un système peut tolérer. Dans la littérature, l'achèvement de la tâche avant un seuil de délai prédéterminé a été considéré comme un paramètre important pour mesurer la qualité de service

(Quality Of Service en anglais (QoS)) du système (Oueis *et al.*, 2015; Ye *et al.*, 2016; Naha et Garg, 2021; Naha *et al.*, 2020). En effet, le seuil du délai de réponse pour un service ou une requête joue un rôle important dans la caractérisation des applications sensibles et intolérantes à la latence. Ce délai devrait être respecté pour un approvisionnement efficace en services et en ressources dans le *fog*.

1.3 Objectifs de la thèse

1.3.1 Objectif général

L'objectif général de cette thèse est de proposer des stratégies pour l'allocation des ressources dans une infrastructure de *fog computing* dynamique. Ces stratégies prennent en considération l'aspect dynamique, largement distribué et très extensible des nœuds du *fog*. Les contrôleurs des différents domaines du *fog* exécutant ces stratégies visent à maximiser le nombre de requêtes dont le délai de réponse ne dépasse pas un seuil prédéterminé tout en équilibrant la charge des différents nœuds et en minimisant les migrations des applications entre les nœuds en temps réel.

1.3.2 Objectifs spécifiques

Les objectifs spécifiques de ma thèse peuvent être résumés comme suit :

— modéliser mathématiquement le problème d'allocation des ressources dans une infrastructure de *fog computing* très dynamique et étudier sa complexité. Cette modélisation permet de fournir des solutions optimales en modélisant le problème comme un programme mathématique puis le résoudre, par exemple, par des algorithmes de séparation et évaluation (Branch and Bound (BnB) en anglais). Ces résultats seront considérés comme des références de comparaison et d'évaluation de performances ;

— proposer des algorithmes sous-optimaux peu-complexes *offlines* pour

l'allocation des ressources qui garantissent théoriquement la qualité des solutions proposées et qui visent à maximiser le nombre de requêtes dont le délai de réponse aux utilisateurs finaux ne dépasse pas un seuil prédéterminé et à distribuer la charge entre les différents nœuds ;

— proposer des stratégies pour l'allocation des ressources en temps réel / *onlines* centralisés et distribués permettant de répondre aux exigences en termes d'extensibilité et de dispersion géographique des nœuds du *fog* ;

— évaluer les performances de ces stratégies en les comparant d'abord à la solution optimale et à d'autres solutions de l'état de l'art, tout en considérant des scénarios réalistes.

1.4 Méthodologie de recherche

Dans ce projet de thèse, nous proposons des stratégies pour l'allocation des ressources dans une infrastructure de *fog computing* dynamique. Pour mener le projet de thèse à sa fin, nous suivons la méthodologie de recherche suivante. D'abord, nous réalisons une recherche bibliographique pertinente de la littérature afin de spécifier les lacunes qui existent dans le cadre d'allocation des ressources dans les environnements de *fog computing* dynamiques. En particulier, nous étudierons les travaux de recherches pour identifier les différentes métriques de performances ainsi que les différentes caractéristiques à considérer. Cette étude bibliographique nous permet de mieux comprendre la problématique, de connaître les approches utilisées ainsi que de cerner les points faibles des solutions existantes. Nous étudierons les solutions *onlines* et *offlines*, centralisées et distribuées.

Après l'étude bibliographique principalement focalisée sur l'optimisation du délai de réponse et de la répartition de charge dans les environnements de *fog computing* dynamiques, nous modélisons mathématiquement le délai de réponse aux requêtes des utilisateurs ainsi que le modèle de répartition de charge. Nous enchaînerons

ensuite avec la modélisation mathématique du problème sous la forme d'un programme d'optimisation où la fonction objectif cherche à maximiser le nombre de requêtes dont le délai ne dépasse pas le seuil prédéfini, tout en équilibrant la charge de travail des nœuds physiques et en respectant les contraintes caractéristiques de l'infrastructure du *fog* ainsi que le dynamisme des nœuds. Nous étudierons ensuite la complexité de ce problème et nous le résolvons d'une manière optimale à l'aide d'outils tels que IBM CPLEX (CPLEX, 2016b). Ces résultats seront considérés comme des références de comparaison et d'évaluation de performances.

Ensuite, nous nous intéresserons à la conception d'algorithmes *offlines* ou méta-heuristiques permettant d'allouer les ressources des nœuds aux applications du *fog* dans le but de maximiser le nombre de requêtes dont le délai ne dépasse pas le seuil prédéfini et de répartir les charges des tâches des nœuds. Nous évaluerons les performances de ces algorithmes par rapport aux solutions optimales obtenues et aux solutions de l'état de l'art. Ces algorithmes peu-complexes devraient aboutir à des solutions tout en ayant les paramètres complets du problème comme entrée. La qualité de ces solutions est garantie théoriquement, mais ne représente pas une solution réaliste étant donné que les paramètres du problème ne peuvent pas être connus depuis le début. Pour remédier à ceci, nous proposerons des algorithmes *onlines*, centralisés et distribués basés sur l'apprentissage machine permettant d'allouer les ressources dans les différents nœuds en tenant compte de leur aspect dynamique, extensible et largement distribué. Nous évaluerons enfin les performances de ces algorithmes par rapport aux solutions précédentes en les implémentant dans des scénarios réalistes.

1.5 Contributions de la thèse

Cette thèse vise à contribuer à l'avancement des connaissances par rapport à une nouvelle technologie qu'est le *fog computing* sur un sujet relativement peu exploré, soit celui de l'allocation des ressources en considérant l'aspect dynamique des nœuds, leur extensibilité et leur distribution géographique.

Par conséquent, nous projettons que cette thèse ait une incidence sur : le développement des orchestrateurs permettant de gérer les nœuds du *fog* ainsi que la conception d'algorithmes pour la gestion des ressources des nœuds du fog. Ces résultats sont, au meilleur de nos connaissances, à la fois nouveaux et uniques. Les travaux de l'état de l'art ont largement étudié le problème d'allocation de ressources dans les environnements de *fog computing*, sans toutefois considérer tous les aspects reliés à la dynamique des nœuds et leur extensibilité. Cette thèse fournit une meilleure compréhension du problème d'allocation des ressources dans les environnements de *fog computing* dynamiques et offre des directives pour les résoudre.

Les différentes contributions de cette thèse sont discutées dans le reste de cette section. Chaque contribution fait l'objet d'une sous-section dans laquelle nous résumons le travail effectué ainsi que les publications scientifiques produites.

1.5.1 Contribution 1 : Algorithmes *offline* pour l'allocation de ressources

Dans une première partie de cette thèse, nous étudions le problème d'allocation de ressources et de provisionnement de tâches dans l'infrastructure de *fog computing* dynamique. Les utilisateurs finaux envoient les tâches des applications qu'ils voudraient effectuer aux couches supérieures pour leur exécution. Le contrôleur du *fog* crée les stratégies d'allocation de ressources et de provisionnement de tâches appropriées pour le déploiement des conteneurs qui contiendront les applications

des utilisateurs. L'objectif de ce problème étant de maximiser le nombre de requêtes dont le délai de réponse est inférieur au seuil prédéfini et d'équilibrer la charge entre les nœuds. Nous formulons d'abord ce problème comme un programme linéaire en nombres entiers. Nous démontrons par la suite que ce problème est NP-Difficile. Puis, nous proposons une méta-heuristique basée sur l'optimisation par essaims particulaires et un algorithme glouton pour la résolution de ce problème. Ces solutions sont évaluées par des simulations sur un système routier intelligent déployé dans un environnement de *fog computing* dynamique et en utilisant des données réelles de mobilité. Nous avons montré que l'algorithme basé sur l'Optimisation à Essaims Particulaires (OEP) obtient des résultats quasi-optimaux, mais avec des temps d'exécution plus longs que l'algorithme glouton.

Ces contributions ont été publiées dans l'article de revue IEEE Journal of Internet of Things 2019 (Mseddi *et al.*, 2019)

1.5.2 Contribution 2 : Algorithme *online* centralisé pour l'allocation de ressources

La deuxième contribution de ce travail a consisté à proposer une solution online pour résoudre le problème de placement de conteneurs exécutant les applications des utilisateurs dans une infrastructure de *fog computing* dynamique. En effet, dans un cadre réaliste, les données à l'entrée du problème ne peuvent pas toutes être disponibles au début de l'exécution de l'algorithme. Nous avons donc commencé par formuler ce problème en tant que programme linéaire en nombres entiers. Puis, nous l'avons résolu. Les résultats obtenus seront considérés comme des références de comparaison et d'évaluation de performances. Nous avons ensuite proposé un algorithme *online* basé sur l'apprentissage machine. Cet algorithme apprendra la dynamique des nœuds du *fog* et des demandes des utilisateurs

afin d'effectuer un placement optimal des conteneurs. Les performances de cet algorithme sont évaluées et les résultats montrent que notre solution atteint des résultats quasi-optimaux en termes de taux de satisfaction, nettement meilleurs que les approches de l'état de l'art.

Ces contributions ont été publiées dans l'article de conférence IEEE Cloudnet 2020 (Mseddi *et al.*, 2020)

1.5.3 Contribution 3 : Algorithme *online* collaboratif pour l'allocation de ressources

Notre troisième contribution a consisté à proposer un algorithme distribué collaboratif et *online*. En effet, l'une des limitations de la solution *online* centralisée, basée sur l'apprentissage machine est que plus la taille du problème augmentait, moins les résultats de la solution étaient performant. En outre, les infrastructures du *fog* sont très extensibles. Et la nature distribuée des nœuds exigeait aussi l'extensibilité et la distribution des contrôleurs qui exécuteront les algorithmes d'allocation de ressources. Nous avons donc proposé une méthode collaborative basée sur l'apprentissage machine, pour résoudre d'une manière distribuée le problème de placement des conteneurs dans l'infrastructure de *fog* dynamique. Puis, nous avons évalué les performances de cette approche et les avons comparées à d'autres approches de l'état de l'art. En outre, nous avons constaté que l'algorithme centralisé est pratique pour les systèmes à petite échelle, tandis que l'algorithme collaboratif est plus adéquat pour les systèmes à plus grande échelle. L'analyse finale a porté sur l'influence de variables clés du système, telles que la fréquence des demandes entrantes, le nombre de nœuds, et la mobilité, sur l'efficacité de la solution collaborative. Cette évaluation a démontré la robustesse de notre approche collaborative, en particulier en ce qui concerne le taux de réussite des opérations dans divers scénarios et configurations du système.

Cette proposition est en cours de ré-évaluation dans la revue IEEE Journal of Internet Of Things.

1.6 Organisation de la thèse

La thèse est organisée comme suit. Nous présentons d'abord une revue critique de la littérature dans le prochain chapitre. Ensuite, nous présentons les différentes contributions de la thèse, chacune dans un chapitre séparé. Le chapitre 3 présente l'architecture de fog adopté ainsi que le modèle considéré et la formulation des problèmes conjoints d'allocation de ressources et de provisionnement des tâches dans un environnement de *fog computing* dynamique. Le chapitre 4 présente une heuristique et une méta-heuristique pour résoudre le problème étudié au chapitre précédent. Dans le chapitre 4, nous étudions le problème de placement de conteneurs et nous présenterons notre méthode online centralisée pour le résoudre. Dans le chapitre 5, nous résolvons le même problème précédent et nous présentons la solution collaborative proposée. Finalement, dans le chapitre 6, nous présentons les conclusions générales, nos travaux en cours, les travaux futurs ainsi que les contributions à l'avancement des connaissances de cette thèse de doctorat.

CHAPITRE II

REVUE DE LITTÉRATURE

2.1 Introduction

Récemment, plusieurs travaux de recherches ont été effectués pour étudier différents défis et problématiques relatifs au *fog computing*. Dans ce qui suit, nous présentons une revue critique des travaux de recherche qui se sont intéressés aux problèmes d'allocation de ressources et de provisionnement des tâches dans les environnements de *fog computing*. Nous étudierons d'abord les travaux qui ont proposé des solutions *offlines*. Puis, nous analyserons les solutions centralisées et distribuées *onlines*.

2.2 Méthodes *offlines* pour l'allocation des ressources

Les plateformes de gestion des ressources dans le *cloud* ne sont pas adaptées à l'environnement de *fog computing* en raison des caractéristiques intrinsèques des nœuds du *fog* comme leur hétérogénéité, leur mobilité et leur variation des ressources. Par conséquent, afin de relever le défi de l'hétérogénéité des ressources, Baker *et al.* ont proposé dans (Baker *et al.*, 2018) une représentation abstraite des ressources et les ont définies de manière à permettre leur découverte, leur composition et leur participation à différents types de scénarios. D'autres travaux ont proposé des plateformes de gestion des ressources adaptées spécifiquement aux

infrastructures de *fog computing*. Dans (Masip-Bruin *et al.*, 2018), les auteurs ont étudié les principales caractéristiques architecturales pour un système de gestion de ressources adapté aux plateformes de *cloud* et de *fog computing*. Ils ont présenté une plateforme de gestion hiérarchique et par couches pour les domaines du *cloud* et du *fog*.

De toute évidence, les appareils mobiles du *fog* introduisent une complexité supplémentaire dans la gestion des ressources. En outre, Jiang et al. (Jiang *et al.*, 2018) ont résumé les principaux défis de l'orchestration de *fog computing*, comme étant l'évolutivité de la couche de contrôle et le transfert des tâches aux nœuds du *fog* suivant leurs préférences au matériel. Ensuite, ils ont discuté de la manière dont le cadre d'orchestration existant du cloud computing pourrait être adapté au cadre du fog computing.

Dans cette section, nous présentons des travaux connexes qui traitent des problèmes d'allocation des ressources et du provisionnement des tâches dans les environnements de *fog computing*, et qui présentent des solutions *offlines*. Nous présentons également une comparaison basée sur plusieurs paramètres des différents travaux étudiés.

2.2.1 Allocation de ressources

Dans (Baburao *et al.*, 2021), Baburao *et al.* ont proposé une méthode d'allocation dynamique des ressources basée sur l'optimisation par essaims de particules, ayant pour objectif de gérer efficacement l'équilibrage de la charge. Cette méthode permet aussi de réduire le temps d'attente des tâches, le délai de réponse et la consommation de bande passante du réseau et ainsi que d'améliorer la qualité de l'expérience (QoE). Leur méthode permet d'allouer les ressources nécessaires en supprimant de la mémoire vive les services inactifs, non référencés et non utilisés depuis longtemps.

Wang et al. (Wang *et al.*, 2017) ont proposé une plateforme de gestion des ressources des nœuds périphériques basée sur des algorithmes de provisionnement et de mise à l'échelle automatique des ressources des nœuds périphériques. Ils ont considéré les variations des requêtes des utilisateurs, mais ont ignoré l'aspect dynamique des nœuds du *fog*.

Dans leur travail (Naha et Garg, 2021), Naha et Garg ont proposé une politique d'allocation de ressources basée sur des critères multiples avec une possibilité de réservation préalable de ressources pour minimiser le délai global, le temps de traitement et les violations de la SLA. Ce processus prend en compte les caractéristiques du *fog computing*, telles que l'hétérogénéité des nœuds, les contraintes de ressources, ainsi que les changements dynamiques des exigences des utilisateurs. Ils utilisent multiples fonctions objectif pour trouver les ressources appropriées pour l'exécution de tâches sensibles à la latence. Cependant, ils n'ont pas pris en compte la mobilité et la variation des ressources disponibles dans les nœuds *fog*. Par ailleurs, Tran-Dang et Kim (Tran-Dang et Kim, 2021b) ont étudié le problème de l'allocation des ressources pour les tâches des applications dans un environnement où les nœuds du *fog* sont des dispositifs hétérogènes complexes, caractérisés essentiellement par des ressources et des capacités de calcul différentes. Ils ont aussi considéré dans leur travail, l'augmentation des taux de demandes de service, qui rendent probablement les files d'attente de tâches dans les nœuds riches en ressources plus longues. En effet, ils ont proposé un algorithme basé sur la priorité des tâches afin de gérer la charge déséquilibrée dans un environnement de *fog computing* hétérogène et d'améliorer le délai de service.

Oueida et al. (Oueida *et al.*, 2018) ont modélisé le processus d'allocation de ressources dans les services d'urgence dans les hôpitaux. Ils ont également proposé un cadre de préservation des ressources utilisant les réseaux de Petri, intégré à des environnements de *cloud* personnalisé et de *fog*, adaptée aux systèmes de services

d'urgence. Leurs résultats de simulation ont illustré des améliorations significatives dans l'utilisation des ressources et la minimisation du temps d'attente des patients.

2.2.2 Provisionnement des tâches

Beaucoup de recherches se sont intéressées au problème du provisionnement de ressources/tâches virtuelles dans les environnements de *fog computing* pour optimiser une fonction objectif. Dans (Skarlat *et al.*, 2016), une heuristique de provisionnement de tâches pour le *fog* est proposée, où les objectifs visent à minimiser le délai de service des requêtes des utilisateurs finaux et à maximiser le taux d'utilisation des ressources dans les nœuds du *fog*. De même, les auteurs de (Hoang et Dang, 2017) ont proposé un algorithme heuristique d'ordonnancement des tâches visant à minimiser le délai de service et à améliorer l'expérience des utilisateurs. Dans (Al-Khafajiy *et al.*, 2018), une plateforme de transfert de tâches vers le *fog computing* basée sur la collaboration des nœuds du *fog* est proposée. L'objectif principal de cette plateforme est de réduire la latence des requêtes.

Le compromis entre la consommation d'énergie et le délai de transmission dans environnement de *fog* et cloud computing est étudié dans (Deng *et al.*, 2016), en équilibrant les charges de travail entre les couches du *fog* et le *cloud*. Les auteurs de (Maamar *et al.*, 2019) ont présenté un modèle de coordination entre le *cloud* et le *fog* pour exécuter des applications IoT en se basant sur les caractéristiques intrinsèques de ces applications. Les caractéristiques étudiées dans leurs travaux, sont la sensibilité des données, leur latence, et leur âge.

Peu de travaux de recherche ont étudié la mobilité et ses conséquences sur les performances du *fog computing*. Waqas et al. (Waqas *et al.*, 2019) fournissent une vue d'ensemble des recherches sur la mobilité des utilisateurs dans le *fog computing*, où les travaux sont classés d'une faible dynamique de l'environnement à une dynamique élevée. Dans les environnements à faible

dynamique, l'environnement de *fog* basé sur les communications appareil à appareil (D2D pour "Device to Device" en anglais) est présentée comme un nouveau paradigme pour le transfert des tâches mobiles vers le *fog* lorsque les utilisateurs finaux se déplacent lentement dans la zone de couverture d'une station de base. De leur part, Lee et *al.* (Lee *et al.*, 2019) ont étudié le problème de la formation dynamique de groupe de nœuds dans le *fog* compte tenu de leurs arrivées et départs imprévisibles causés par leur mobilité. Dans le cadre d'une architecture hybride cloud-fog, les auteurs visent à minimiser la latence maximale de communication et de calcul en concevant conjointement des groupes de nœuds et en optimisant la répartition des tâches entre eux.

Sookhak et *al.* (Sookhak *et al.*, 2017) ont proposé d'exploiter les véhicules en stationnement comme nœuds du *fog* afin d'augmenter les ressources de calcul dans l'environnement de *fog computing*. Ils ont également décrit une architecture de l'infrastructure du *fog computing véhiculaire* (FCV) proposée et élucidé le rôle des composants constitutifs de sa plateforme de gestion des ressources. Les auteurs n'ont cependant pas considéré les voitures en déplacement dans leur étude. Ainsi, c'est seulement la disponibilité d'un nœud à un moment donné qui est prise en compte. Chen et *al.* (Chen *et al.*, 2017) ont proposé de minimiser la consommation d'énergie en utilisant une approche hybride de provisionnement des tâches, où les utilisateurs finaux peuvent choisir entre différents modes de transfert des tâches. Ils ont défini un environnement hautement dynamique de fog computing véhiculaire. Dans leur travail, les véhicules sont, en général, les générateurs de données, et les unités de bord de route (RSU pour RoadSide Unit en anglais) représentent les nœuds du fog. Dans (Hou *et al.*, 2016), les auteurs ont exploité des véhicules en stationnement pour effectuer des tâches de *fog*. Cependant, lorsque les véhicules se déplacent, les auteurs ont justifié qu'ils ne peuvent servir que de nœuds de communication opportunistes, tandis que les tâches transférées sont traitées

par des nœuds du *fog* stationnaires. L'utilisation de véhicules en mouvement comme nœuds du *fog* dans FCV a été étudiée dans (Wang *et al.*, 2018b; Zhu *et al.*, 2019). Dans (Wang *et al.*, 2018b), une conception détaillée de la façon d'utiliser les véhicules stationnés et en mouvement conjointement avec les RSUs est présentée. En outre, les auteurs de (Zhu *et al.*, 2019) ont proposé un algorithme de provisionnement de tâches optimisé pour la minimisation de la latence dans les FCV. De plus, la mobilité n'est pas directement prise en compte dans leur étude, mais plutôt reflétée par la présence/absence d'un nœud dans une zone étudiée à un instant donné. D'autre part, (Wang *et al.*, 2018b) limite le déplacement des nœuds du *fog* aux véhicules présents dans la portée de communication des RSUs. De plus, aucun des travaux susmentionnés n'a pris en compte la disponibilité irrégulière des ressources au sein des nœuds du *fog*.

À notre connaissance, notre étude portant sur l'allocation des ressources dans un environnement de *fog computing* dynamique est le premier travail où la mobilité et la disponibilité sporadique des ressources aux nœuds du *fog* sont prises en compte. Finalement, dans (Mseddi *et al.*, 2019), nous avons étudié le problème conjoint du placement des conteneurs et du provisionnement des tâches dans un environnement de *fog computing* dynamique. Nous avons proposé une méta-heuristique peu complexe basée sur la méthode d'optimisation par essaims particulaire (OEP ou PSO Particle Swarm Optimization en anglais) et une heuristique gloutonne visant à maximiser le nombre d'utilisateurs finaux servit tout en tenant compte du comportement et de la mobilité dynamiques des nœuds du *fog* et de la disponibilité des ressources.

2.2.3 Méthodes heuristiques décentralisées

Comme étudiées dans (Yang *et al.*, 2021), les collisions peuvent se produire lorsque plusieurs utilisateurs transfèrent leurs tâches au même nœud. Les auteurs

proposent alors dans leur étude un accord pour la résolution des collisions. L'algorithme proposé peut fonctionner de manière complètement décentralisée, chaque utilisateur effectuant sa propre observation et prenant sa propre décision. D'autre part, Lyu *et al.* ont proposé dans (Lyu *et al.*, 2018) un modèle de transfert des taches vers le *fog* entièrement distribué, visant à minimiser le temps moyen des groupes de nœuds pour un calcul collaboratif. En créant des régions de calcul collaboratif à proximité des utilisateurs, les auteurs ont illustré la capacité d'empêcher le transfert de tâches au-delà de ces régions, permettant ainsi de réduire le délai de service. Finalement, les auteurs de (Gao *et al.*, 2020) ont étudié le problème du transfert de tâches en temps réel et du compromis entre la puissance consommée et le délai de service. Les auteurs ont proposé un modèle de transfert de tâches et d'allocation de ressources distribué et prédictif pour les systèmes FC, montrant la quasi-optimalité des économies d'énergie et la stabilité des files d'attente , grâce au développement d'un modèle de file d'attente optimisée.

Les travaux de l'état de l'art étudiés dans cette section sont présentés dans le tableau 2.1 en fonction de leurs objectifs et caractéristiques. Les objectifs de ces travaux peuvent se résumer en la minimisation des délais, l'économie d'énergie et l'efficacité de l'utilisation des ressources. Les solutions de gestion des ressources et de provisionnement des tâches peuvent être classées en fonction des caractéristiques considérées, telles que la mobilité des utilisateurs (Us) et des nœuds du *fog* (NFs), la localisation des NFs, la variation des ressources dans le temps et les récompenses accordées aux NFs suite à leurs contribution au *fog*. Certains autres travaux ont considéré que les tâches ne peuvent être attribuées qu'aux NFs compatibles, ou que les NFs peuvent collaborer dans l'exécution des tâches du *fog* . Certains travaux ont juste considéré les problèmes liés à l'allocation des ressources (AR), alors que d'autres ont considéré les problèmes liés au transfert des tâches (TT).

Les travaux présentés dans cette section ont proposé des solutions heuristiques peu complexes, mais qui atteignent des performances quasi-optimales. Cependant, l'environnement de *fog* est caractérisé par sa haute dynamique. Ainsi, les paramètres de la mobilité et de variation de ressources des nœuds du fog ainsi que la variation de la charge des requêtes des utilisateurs ainsi que leur mobilité ne peuvent pas être connus depuis le début. Ces solutions ne représentent donc pas des solutions réalistes. Par conséquent, des algorithmes *onlines* d'apprentissage automatique plus complexes, ont été utilisés pour atteindre de meilleures performances pour les problèmes d'allocation des ressources dans le *fog*.

Table 2.1 Comparaison des solutions *offlines* pour l'allocation des ressources

Papier	Intérêt	Métriques de performance			Caractéristiques						
		Délai	Énergie	Usage des ressources	Mobilité		Localité des NFs	Variation des resources	Récomp.	Compatib. NF / tâche	Collab. NFs
					Us	NFs					
(Baburao *et al.*, 2021)	AR	✓	—	—	✓	—	—	—	—	—	✓
(Naha et Garg, 2021)	AR	✓	—	—	✓	—	—	✓	✓	—	—
(Tran-Dang et Kim, 2021b)	TT	✓	—	✓	—	—	✓	—	—	✓	—
(Lee *et al.*, 2019)	TT- AR	✓	—	—	—	—	✓	✓	—	✓	✓
(Skarlat *et al.*, 2016)	AR/TT	✓	—	✓	—	—	—	—	—	✓	✓
(Sookhak *et al.*, 2017)	AR- TT	✓	—	—	—	—	✓	✓	✓	—	—
(Masip-Bruin *et al.*, 2018)	AR	✓	—	—	✓	—	—	—	—	✓	—
(Wang *et al.*, 2017)	AR	✓	—	✓	—	—	—	✓	—	—	—
(Oueida *et al.*, 2018)	AR	✓	—	✓	—	—	—	—	—	—	—

Papier	Intérêt	Métriques de performance			Caractéristiques					
		Délai	Énergie	Usage des ressources	Mobilité		Localité des NFs	Variation des resources	Récomp.	Compatib. NF / tâche	Collab. NFs
					Us	NFs					
(Al-Khafajiy *et al.*, 2018)	TT	✓	—	—	—	—	✓	✓	—	—	✓
(Wang *et al.*, 2018b)	TT	✓	—	—	—	—	✓	—	—	—	—
(Zhu *et al.*, 2019)	TT	✓	—	—	—	✓	✓	—	—	—	—
(Hoang et Dang, 2017)	TT	✓	—	—	—	—	✓	—	—	—	—
(Deng *et al.*, 2016)	TT	✓	✓	—	✓	—	—	—	—	—	—
(Maamar *et al.*, 2019)	TT	✓	—	—	—	—	—	—	—	✓	—
(Pu *et al.*, 2016)	TT	—	✓	—	✓	—	✓	—	✓	—	—
(Chen *et al.*, 2017)	TT	—	✓	—	—	—	✓	—	✓	—	—
(Hou *et al.*, 2016)	TT	✓	—	✓	—	—	✓	—	—	—	—
(Mseddi *et al.*, 2019)	AR - TT	✓	—	✓	✓	✓	✓	✓	—	—	—

2.3 Méthodes *onlines* basées sur l'apprentissage machine

Les solutions *onlines* basées sur l'apprentissage machine ont été largement étudiées. Ces solutions ont été principalement développées pour le *cloud computing* (Salahuddin *et al.*, 2016; Cui *et al.*, 2018; Bitsakos *et al.*, 2018; Zhang *et al.*, 2017), et ce n'est que récemment que leur utilisation dans les plateformes de *fog computing* a été explorée (Tran-Dang *et al.*, 2022). En outre, la plupart des solutions proposées ont ignoré l'hétérogénéité des nœuds du *fog* en termes de mobilité et de variations de la charge de travail. Ces méthodes d'apprentissage machine peuvent être divisées en méthodes centralisées où une unique entité centrale exécute l'algorithme d'apprentissage pour résoudre le problème étudié, et des algorithmes distribués où un ensemble d'agents d'apprentissage collaborent ou rivalisent pour résoudre le problème. Dans ce qui suit, nous étudierons des travaux de l'état de l'art qui ont proposé des solutions en utilisant ces méthodes d'apprentissage machine centralisées et distribuées.

2.3.1 Méthodes centralisées basées sur l'apprentissage machine

Récemment, Talaat *et al.*.ont présenté une nouvelle méthodologie pour la prédiction et l'allocation de ressources adaptée aux applications de santé, pour les environnements de *fog computing* (Talaat, 2022). Les auteurs proposent deux modules. D'abord, un module pour l'allocation des ressources qui apprend à sélectionner le meilleur nœud pour exécuter les requêtes des utilisateurs en utilisant un algorithme d'apprentissage par renforcement (RL pour "Reinforcement Learning" en anglais). Ce module a pour but d'équilibrer la charge entre les différents nœuds de l'infrastructure de *fog*. Le deuxième module de prédiction utilise le réseau neuronal probabiliste pour prédire les demandes futures des utilisateurs .

Dans (Zhu *et al.*, 2018), Zhu *et al.* ont étudié le problème de transfert des tâches de calcul vers le *fog*, où les utilisateurs finaux sont mobiles et leurs emplacements sont inconnus des nœuds du *fog*. Le problème associé est formulé comme un jeu de bandits à plusieurs bras ("multi-armed bandit" en anglais) et résolu à l'aide d'une approche d'apprentissage par renforcement. Les résultats analytiques et de simulation ont validé l'efficacité de leur méthode. En outre, Rejiba *et al.* ont aussi étudié dans (Rejiba *et al.*, 2019) le problème de transfert de calcul dans un système de *fog computing véhiculaire*. Grâce à l'apprentissage basé sur le problème du bandit à bras multiples, le délai de service a été amélioré. De plus, Wang *et al.* ont étudié le problème conjoint de l'allocation des tâches en ligne et de l'ordonnancement du spectre dans un environnement mobile de *fog computing* (Wang *et al.*, 2019a). Ils ont proposé une solution *online* basée sur l'algorithme des bandits à bras multiples, où les utilisateurs et les nœuds du *fog* sont mobiles. Ils ont cherché à minimiser la latence du transfert des tâches.

L'allocation des ressources pour les différentes tâches du *fog* est couplée les unes aux autres et les informations futures ne peuvent être obtenu. Comme la fréquence d'arrivée des tâches à traiter par le *fog* est variable, les auteurs de (Fan *et al.*, 2022) ont conçu un algorithme d'apprentissage par renforcement tenant compte des délais de réponse pour obtenir une solution sous-optimale et en temps réel pour l'allocation des ressources radio et de calcul pour le traitement des tâches ; sur la base des données de relecture de l'expérience du système.

Chan *et al.* ont exploré l'équilibrage des charges de calculs et de communications pour les environnements de *fog computing*, à la volée, pour minimiser la latence de service (Chen *et al.*, 2018). En raison de l'hétérogénéité des nœuds du *fog*, les décisions doivent s'adapter de manière flexible aux demandes imprévisibles des utilisateurs et à la disponibilité des ressources des nœuds.

Wang *et al.* ont exploité le Q-learning et le Deep Q-learning pour le transfert de trafic et de calcul dans un environnement FCV basé sur les communications machine à machine (D2D pour Device To Device en anglais) (Wang *et al.*, 2019b). Par le biais de simulations et sur la base de plusieurs traces de trajectoires mobiles, ils ont montré que leurs approches surpassent les méthodes de référence en termes de coût énergétique et de délai de service. Dans (Tang *et al.*, 2019), Tang *et al.* ont étudié le déploiement et la migration des conteneurs de *fog* tout en prenant en charge les utilisateurs mobiles et hétérogènes. Les auteurs ont proposé un nouveau gestionnaire de migration des conteneurs basé sur l'apprentissage par renforcement profond pour réduire le délai, la consommation d'énergie et le coût de la migration.

En outre, Lee *et al.* ont proposé dans (Lee et Lee, 2020) une solution heuristique améliorée par un algorithme d'apprentissage par renforcement pour l'allocation des ressources dans la FCV. Les auteurs ont étudié dans (Lee et Lee, 2020) le problème de l'allocation des ressources limitées des nœuds pour le FCV. Ils ont aussi utilisé des informations de prédiction sur le mouvement des véhicules et leur statut de stationnement, collectée à partir de l'environnement intelligent de la ville. Les résultats de la simulation ont montré que cette approche permettait d'obtenir une plus grande qualité de service par rapport aux algorithmes classiques d'allocation des ressources.

Dans (Ning *et al.*, 2019a), Ning *et al.* ont construit une plateforme de transfert de calcul à trois couches dans un environnement de FCV pour minimiser la consommation d'énergie, tout en satisfaisant les contraintes de délai des utilisateurs. Le problème est formulé comme une combinaison de redirection de flux et d'allocation de tâches. Le problème de la redirection du flux est résolu à l'aide de l'algorithme d'Edmonds-Karp et le problème d'allocation de ressources à l'aide d'un algorithme d'apprentissage par renforcement profond.

Dans (Rahman *et al.*, 2020), Rahman *et al.* proposent un schéma d'allocation des ressources et de transfert des calculs basé sur l'apprentissage par renforcement profond (DRL) qui permet d'obtenir une solution sous-optimale dans les réseaux d'accès radio du fog (FRAN pour fog radio access network en anglais). L'idée de la proposition est que le contrôleur DRL décide intelligemment s'il faut traiter la tâche de calcul générée localement au niveau du dispositif ou la transférer à un point d'accès du *fog* ou à un serveur du *cloud* et alloue une quantité optimale de ressources de calcul et de puissance en fonction du service demandé. Cependant, les auteurs n'ont pas pris en compte le comportement dynamique des nœuds du fog.

Dans (Fang *et al.*, 2022), les auteurs ont proposé un schéma d'allocation de ressources basé sur l'apprentissage par renforcement profond (DRL) pour améliorer la distribution de contenu dans un FRAN. Ils ont formulé le problème d'allocation des ressources comme un modèle de délai minimal, où la mise en cache dans le réseau est déployée et où les mêmes demandes de contenu provenant d'utilisateurs mobiles peuvent être regroupées dans la file d'attente de chaque station de base. Dans leurs solution, une nouvelle politique DRL est conçue pour prendre des décisions de mise en cache et de routage coopératifs pour les demandes des utilisateurs, en fonction de l'information sur l'historique des demandes et des ressources réseau disponibles dans le système.

Finalement, nous avons présenté dans (Mseddi *et al.*, 2020) un modéle de FC où le comportement dynamique des utilisateurs et des NFs est pris en compte pour l'allocation des ressources virtualisées. Le RL centralisé a été exploité pour le déploiement de conteneurs en ligne et l'approvisionnement en services. Des simulations ont démontré que notre méthode atteint une performance quasi-optimale en termes de nombre de demandes FC satisfaites.

2.3.2 Méthodes distribuées basées sur l'apprentissage machine

Les méthodes basées sur le RL où plusieurs agents collaborent pour résoudre les problèmes d'allocation des ressources ont d'abord été proposés pour les environnement de *Cloud Computing* (Yelina *et al.*, 2020) où leur performances était remarquable. Récemment, des travaux ont aussi investigué l'utilisation de ces méthodes pour résoudre les problèmes d'allocation des ressources dans le *fog computing*

En revanche, Zhao *et al.* ont proposé dans (Zhao *et al.*, 2020) un nouveau mécanisme basé sur la récompenses pour inciter les nœuds à collaborer dans le fog. Leur méthode est basée sur un modèle de contrats pour le transfert de tâches dans les systèmes FCV. En utilisant un algorithme DRL distribué, l'allocation des ressources est optimisée. Les résultats numériques ont démontré l'efficacité du schéma proposé en matière de transfert de tâches et d'allocation de ressources. De plus, Alam *et al.* ont proposé dans (Alam *et al.*, 2016) un modèle distribué de transfert de tâches basé sur RL pour minimiser la latence dans un système FC mobile et en temps réel.

L'hétérogénéité des nœuds du *fog* quant à leurs capacités de calcul peut créer un long retard dans l'exécution des tâches à cause des longues files d'attente qui peuvent être créées dans certains nœuds. Pour gérer le conflit des différentes demandes pour les ressources pour le traitement des tâches, l'article (Tran-Dang et Kim, 2021a) propose un algorithme distribué d'allocation des ressources dans le *fog*, à savoir MaxRU (Maximum Resource Allocation en anglais). En raison du manque d'informations globales, les nœuds du fog prennent des décisions sur les demandes de tâches qui sont acceptées pour être traitées de manière distribuée. Dans l'algorithme MaxRU, certaines des demandes reçues sont acceptées pour être servies par un certain nœud de sorte que l'utilisation des ressources du *fog* soit

maximisée.

Par ailleurs, Lieu *et al.* étudient le mécanisme de transfert des tâches de calcul de multiples utilisateurs "égoïstes" conjointement avec le problème d'allocation des ressources dans les réseaux de périphérie (Edge Networks en anglais) en formulant un jeu stochastique (Liu *et al.*, 2020). Dans ce jeu, chaque utilisateur est un agent d'apprentissage qui observe son environnement local pour apprendre les décisions optimales dans le but de minimiser le coût du système à long terme en choisissant son niveau de puissance d'émission, sa RAT et son sous-canal sans connaître aucune information sur les autres utilisateurs. Puisque les décisions des utilisateurs sont couplées à la passerelle, les auteurs définissent la fonction de récompense de chaque utilisateur en considérant l'effet agrégé des autres utilisateurs. Par conséquent, un cadre d'apprentissage par renforcement multi-agent est développé pour résoudre le jeu avec l'algorithme d'apprentissage *Q-Learning* basé sur des apprenants indépendants.

Dans l'article (Cheng *et al.*, 2022), les auteurs ont étudié conjointement les problèmes de partitionnement des tâches et de contrôle de la puissance dans un réseau de *fog computing* comportant des nœuds mobiles et non mobiles. Chaque tâche peut être partitionnée en plusieurs sous-tâches transférées vers les nœuds du *fog* selon la stratégie de partition des tâches et la stratégie de puissance de transmission proposées afin de réduire le délai d'exécution des tâches et la consommation d'énergie. À cette fin, les auteurs présentent un algorithme de transfert de tâches vers les nœuds du *fog*, basé sur le gradient de politique déterministe profonde multi-agent (MADDPG pour Multi-Agent Deep Deterministic Policy Gradient en anglais) afin de maximiser l'utilité du système à long terme, y compris le délai d'exécution et la consommation d'énergie.

Table 2.2 Comparaison des solutions *onlines* pour l'allocation des ressources

Papier	**Hypothèses**						**Objectifs et solutions proposées**
	Mobilité	**Ressources**	**Collaboration**	**Type de réseau**	**Requêtes**	**Autre**	
(Talaat, 2022)	Non	Statiques	Non	Réseau sans fil	Variables	-	Méthodologie pour la prédiction et l'allocation des ressources adaptée aux applications de santé
(Fan *et al.*, 2022)	UEs mobiles	Statiques	Non	Réseau sans fil	Variables	-	Allocation de ressources basée sur l'apprentissage par renforcement profond pour la distribution de contenu dans les réseaux du *fog*.
(Xu *et al.*, 2018)	FNs Mobiles	Statiques	Non	Réseau véhiculaire	Statique	–	Plateforme FC véhiculaire qui minimise le retard du réseau en optimisant l'affectation des tâches entre les UE et les FNs véhiculaires.
(Tran-Dang et Kim, 2021a)	UEs mobiles	Variables	Non	Réseau sans fil	Variables	-	algorithme d'allocation des ressources (ressources radio et les ressources de calcul) à la fois dans le canal sans fil et dans les NFs pour minimiser le délai
(Ning *et al.*, 2019b)	Voitures mobiles et stationné	Statiques	Oui	Réseau véhiculaire	Variable	–	Modèle de FC véhiculaire qui minimise le délai de réponse du réseau en fonction de l'emplacement.

Papier	**Hypothèses**						**Objectifs et solution proposée**
	Mobilité	**Ressources**	**Collaboration**	**Type de réseau**	**Requêtes**	**Autre**	
(Lyu *et al.*, 2018)	Non	Variable	FNs dans la même region	Différents délais des lliens	Variable borné par une valeur maximale	Coût des ressources	Transfert distribué de tâches d'une manière en ligne vers des régions collaboratives dans une infrastructure Fog de grande échelle afin de minimiser le temps de réponse.
(Gao *et al.*, 2020)	Non	Statique	Non	Réseau Cloud-fog	Predit	Consommation d'énergie des FNs	Allocation dynamique des ressources et transfert de tâches avec prédiction pour la stabilité du trafic et une consommation minimale d'énergie.
(Wang *et al.*, 2017)	UEs Mobiles	Statiques	Non	–	Statique	Coût des resources	Méthode en ligne pour l'allocation des ressources fog-cloud en vue de minimiser le coût de transfert des tâches.
(Wu *et al.*, 2020)	Arrivée et départ des voitures	Statiques	Non	Réseau véhiculaire (802.11p)	Variable	–	Méthode en temps réel pour le déchargement des tâches au FC pour maximiser la récompense à long terme comme fonction du délai de transmission, du délai de calcul, des ressources disponibles et de la diversité des véhicules et des tâches.

Papier	Hypothèses						Objectifs et solution proposée
	Mobilité	Ressources	Collaboration	Type de réseau	Requêtes	Autre	
(Yang *et al.*, 2021)	Non	Variables	Non	Réseau sans fil	Variable	–	Politiques décentralisées de transfert de tâches en temps réel où chaque utilisateur décide du FN où sa tâche sera transférée.
(Chen *et al.*, 2018)	Non	Statique	Oui	Réseau Cloud-fog	Variable	Experience Utilisateur	Transfert de tâches en ligne au niveau d'un réseau fog/cloud pour minimiser la latence du service.
(Zhu *et al.*, 2018)	UEs Mobiles	Statiques	Non	Réseau sans fil	Variable	–	Transfert de tâches dans le FC avec une mobilité des UEs inconnue en vue de minimiser la latence et de satisfaire la qualité d'expérience.
(Chen *et al.*, 2019)	FNs mobiles (même vitesse)	Statique	Non	Réseau sans fil	Statique	–	Algorithme basé sur le DRL qui minimise le temps de calcul, délai de transfert des tâches et délai de transfert des paquets.
(Wang *et al.*, 2019b)	FNs Mobiles	Statiques	Non	Réseau véhiculaire basé sur D2D	Variable	–	Transfert de trafic et de tâches basé sur RL pour minimiser les délais de service et les coûts énergétiques.

Papier	**Hypothèses**						**Objectifs et solution proposée**
	Mobilité	**Ressources**	**Collaboration**	**Type de réseau**	**Requêtes**	**Autre**	
(Tang *et al.*, 2019)	UEs Mobiles	Statiques	Non	Réseau sans fil	Statique	Consommation d'énergie	Déploiement et migration de conteneurs basés sur le RL pour minimiser les délais, l'énergie et les coûts de migration.
(Wang *et al.*, 2019a)	UEs et FNs Mobiles	Statique	Non	Réseau sans fil	Statique	–	Algorithme basé sur le bandit à bras multiples pour le déploiement des conteneurset leur migration pour minimiser les délais, l'énergie et les coûts de migration.
(Rejiba *et al.*, 2019)	FNs Mobiles	Variable	Non	Réseau Véhiculaire	Statique	–	Affectation en ligne des tâches dans un système de communication véhiculaire par le biais de conseils des nœuds voisin afin de minimiser les délais de service.
(Ning *et al.*, 2019a)	Voitures stationnées et mobiles	Statique	Non	Réseau Véhiculaire	Variable	Contrainte de délai	Algorithme basé sur Edmonds-Karp pour la redirection des flux des UEs et basé sur le RL pour l'allocation des ressources en vue de minimiser la consommation d'énergie.
(Lee et Lee, 2020)	Voitures stationnées	Variable	Non	Réseau véhiculaire	Variable	–	Allocation de ressources en temps réel , basée sur le RL combiné à une heuristique pour minimiser le délai de résponse.

Papier	Hypothèses						Objectifs et solution proposée
	Mobilité	**Ressources**	**Collaboration**	**Type de réseau**	**Requêtes**	**Autre**	
(Mseddi *et al.*, 2020)	UEs et FNs Mobiles	Variable	Non	Réseau sans fil	Variable	–	Provisionnement des tâches et placement des conteneurs en temps réel basés sur le RL en vue de maximiser le taux de réussites.
Notre 3^{me} contribution	UEs and FNs Mobiles	Variable	Non	Réseau sans fil	Variable	–	Placement des conteneurs en temps réel basé du RL distribué en vue de maximiser le taux de réussites.

2.4 Limites des solutions de l'état de l'art

Le *fog computing* est une infrastructure caractérisée par la dynamique de ses nœuds en termes de mobilité, disponibilités, variations des capacités de ressources, énergie... D'autre part, les utilisateurs finaux qui interagissent avec cette infrastructure sont tout aussi dynamiques et hétérogènes. Les solutions d'allocation de ressources et de transferts de tâches conçues pour cet environnement devraient tenir compte de cette dynamique. Cependant, nous pouvons conclure après l'étude des travaux pertinents que souvent seulement quelques paramètres caractérisant l'environnement du *fog* sont considérés. En outre, nous pouvons constater que certains de ces travaux comme (Zhu *et al.*, 2018; Hoang et Dang, 2017; Deng *et al.*, 2016; Chen *et al.*, 2018) n'ont pas considéré la variation de la charge des nœuds. D'autre part, nous pouvons voir que plusieurs travaux (Skarlat *et al.*, 2016; Masip-Bruin *et al.*, 2018; Wang *et al.*, 2017; Wang *et al.*, 2019a; Maamar *et al.*, 2019) n'ont pas pris en considération les nœuds du *fog*. D'autres travaux (Zhu *et al.*, 2019) ont considéré la mobilité des nœuds du *fog* mais pas celle des utilisateurs. Ainsi, nous pouvons conclure que bien que ces travaux considèrent principalement la minimisation du délai de service ou de l'énergie, les caractéristiques des environnements sont limitées. Les travaux ayant proposé des solutions *offlines* ont aussi les mêmes limitations. En outre, ces méthodes ont souvent considéré la mobilité des nœuds du fog surtout avec l'avènement du *fog computing véhiculaire* où les nœuds sont souvent mobiles. Certains travaux (Lee et Lee, 2020) ont aussi considéré uniquement les voitures stationnées pour l'allocation des ressources. Cependant bien que les aspects dynamiques des nœuds ont été mieux considérés dans les travaux adoptant les méthodes *onlines*, nos contributions ont couvert plus d'aspects de dynamisme.

2.5 Conclusion

Dans ce chapitre, nous avons présenté une revue critique des travaux de recherche qui se sont intéressés aux problèmes d'allocation de ressources et de provisionnement des tâches dans les environnements de *fog computing*. Nous avons d'abord étudié les travaux qui ont proposé des solutions *offlines*. Par la suite, nous avons analysé les solutions centralisées et distribuées *onlines*. Cependant, nous pouvons conclure après l'étude de ces travaux que seulement quelques paramètres caractérisant le dynamisme des nœuds du *fog* ont été considérés dans la littrrature.

CHAPITRE III

ARCHITECTURE ET MODÈLE

3.1 Introduction

Nous commençons ce chapitre par présenter l'architecture de *fog computing* considérée dans notre travail, ainsi que ses différents modules. Ensuite, nous modélisons l'infrastructure du *fog* dynamique basée sur les conteneurs comme moyen de virtualisation permettant de déployer les applications des utilisateurs finaux. Enfin, le problème conjoint de placement de conteneurs et de provisionnement de tâches est formulé comme un ILP.

3.2 La plateforme de *fog computing*

Comme le montre la Figure 3.1, nous considérons une architecture de *fog computing*, composée de trois couches. La couche la plus basse est constituée des appareils des utilisateur/IoT, qui peuvent être des smartphones, des tablettes, des appareils portables, des nœuds de capteurs sans fil, etc. Ces appareils envoient des demandes aux couches supérieures pour l'exécution des applications.

La couche intermédiaire représente l'environnement du *fog computing* où les calculs sont effectués. Elle est formée par plusieurs domaines qui peuvent être délimités sur la base de la région géographique ou de la connectivité réseau (Masip-Bruin *et al.*, 2018). Chaque domaine est géré par un contrôleur désigné.

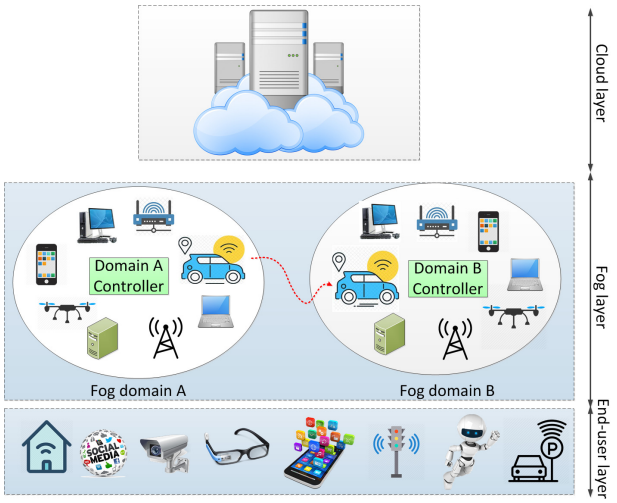


FIGURE 3.1 Architecture de *fog computing*

Les nœuds du FC sont des dispositifs intelligents, tels que les routeurs présents à la périphérie du réseau, les commutateurs, les points d'accès, les ordinateurs portables, les téléphones intelligents, les véhicules, etc. Ces appareils disposent de capacités de calcul, de stockage de données et de communications. Finalement, la couche supérieure représente les centres de données du cloud computing, où l'exécution des tâches et le stockage à long terme sont effectués.

Il est important de rappeler que les ressources considérées peuvent référer à toute composante dans un nœud du *fog* pouvant être accessible et géré par le contrôleur du *fog* à travers une interface. Ceci pourrait inclure la mémoire vive, le processeur, le réseau, les micro-services, les machines virtuelles, les conteneurs, etc. (De Brito *et al.*, 2017).

Afin d'assurer l'isolement des applications et de garantir la sécurité de la plateforme de fog computing, les techniques de virtualisation des ressources telles que les machines virtuelles et les conteneurs représentent les outils de déploiement d'applications privilégiés (Santo *et al.*, 2019). Comme les nœuds

du *fog* sont des appareils ayant des ressources limitées, les conteneurs légers comme LXC (Helsley, 2009) et Docker (Merkel, 2014) sont considérés comme des approches prometteuses pour gérer les tâches des applications (Prateeksha et Yogesh, 2017; Yannuzzi *et al.*, 2014). Par conséquent, chaque nœud doit contenir une plateforme de conteneurisation telles que Docker, LXC, etc., pour gérer localement ses conteneurs.

Les utilisateurs finaux envoient les tâches des applications qu'ils voudraient effectuer aux couches supérieures pour exécution. Dans la couche du *fog*, un contrôleur du *fog* crée la stratégie d'allocation de ressources appropriée pour le déploiement des conteneurs qui contiendront les applications des utilisateurs, afin de satisfaire un nombre maximum de demandes d'utilisateurs tout en tenant compte des emplacements de ces utilisateurs ainsi que celles des nœuds du *fog*. Le contrôleur du *fog* tient en compte aussi des ressources disponibles dans les nœuds et des exigences en termes de délais des applications à déployer. Il envoie les demandes de déploiement de conteneurs au nœud où les tâches de leurs applications doivent être exécutées et redirige les utilisateurs vers eux. Afin d'utiliser efficacement les ressources des nœuds du *fog*, un conteneur peut exécuter des tâches de plusieurs utilisateurs pour une seule application. Le contrôleur de domaine du *fog* surveille le délai de réponse des applications des utilisateurs et prend des décisions de migration de conteneurs afin d'optimiser le nombre d'utilisateurs satisfaits. Les migrations sont effectuées par les plateformes de conteneurisation embarquées dans chaque nœud.

3.2.1 Le contrôleur du domaine du *fog*

Dans la couche du *fog*, un nœud désigné dans chaque domaine fait office de gestionnaire de ressources, appelé contrôleur de domaine. Ce contrôleur

1. reçoit les requêtes/tâches des utilisateurs ;

2. gère les ressources disponibles sur les nœuds du *fog*, par exemple la capacité de calcul, la bande passante du réseau, etc. ; et par conséquent
3. crée la stratégie d'allocation des ressources la plus appropriée pour les applications des utilisateurs.

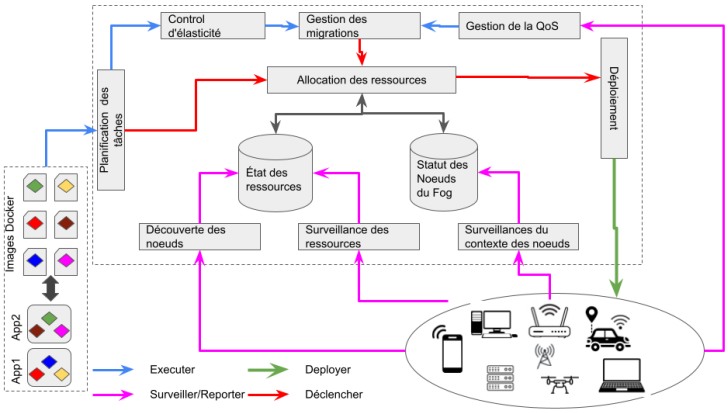


FIGURE 3.2 Architecture du contrôleur de domaine

Dans la Figure 3.2, nous illustrons l'architecture proposée du contrôleur de domaine, inspirée des architectures de contrôleur de nuage présentées dans (Manvi et Shyam, 2014). Cette architecture est composée des modules suivants :

— *Découverte de nœuds* : Ce module détermine quels nœuds du *fog* se trouvent dans le domaine du contrôleur. Il est responsable du suivi des entrées et sorties dynamiques des nœuds dans le domaine contrôlé.

— *Surveillance des ressources* : Ce module surveille périodiquement l'état de disponibilité des ressources matérielles de calcul et de mise en réseau des nœuds. Il doit avoir une connaissance précise des ressources de chaque nnœud. En général, l'information recueillie est stockée dans le *référentiel de ressources.*

— *Surveillance de contexte* : Ce module suit à la fois la mobilité et les variations des ressources de tous les nœuds du domaine. En effet, le contrôleur doit être capable d'anticiper l'état futur d'un nœud afin de placer

adéquatement les conteneurs et bien distribuer les tâches. L'intégration de ce module est une nouveauté dans ce travail, propre à l'environnement dynamique du fog computing.

— *Planification des tâches* : Ce module reçoit les requêtes des utilisateurs indiquant les tâches des applications à exécuter au niveau de la couche de *fog*. Une requête devrait comprendre tous les paramètres et informations pertinents tels que l'identifiant de l'image de l'application, le niveau de qualité de service requis, la charge de travail de chaque tâche et la quantité de données. Ce module décide si la requête de l'utilisateur peut être exécutée sur un conteneur déjà déployé ou si la création d'un nouveau conteneur est nécessaire pour répondre aux exigences de QoS.

— *Allocations des ressources* : Ce module est responsable d'affecter les ressources des nœuds du *fog* aux conteneurs. Pour ce faire, il prend compte des exigences de qualité de service de l'application, les demandes actuelles des utilisateurs et de la disponibilité des ressources.

— *Gestion des migrations* : Ce module reçoit et satisfait les demandes de migration de conteneurs. Il interagit avec les plateformes de conteneurisation des nœuds du domaine afin d'exécuter le processus de migration des conteneurs.

— *Contrôle d'élasticité* : Ce module est responsable de la mise à l'échelle des conteneurs pour chaque application en fonction du nombre de demandes des utilisateurs, en allouant plus ou moins de ressources à ces conteneurs.

— *Gestion de la QoS* : Ce module garantit que les nœuds du fog peuvent maintenir le niveau de QoS requis par les requêtes des utilisateurs. Par exemple, le seuil sur le délai de service peut être défini comme une contrainte par exemple. Si la QoS n'est pas respectée, ce module peut interagir avec le module de migration afin de changer l'emplacement du conteneur.

— *Déploiement* : Ce module interagit avec le référentiel d'images d'applications dans le *cloud* afin de déployer les conteneurs dans les nœuds avec les ressources requises.
— *Registre des ressources* : Ce module stocke toutes les informations sur les ressources physiques de tous les nœuds du domaine. Il collecte ses données à partir du module *Découverte des nœuds*.
— *Registre d'états* : Ce module stocke les modèles de mobilité et les modèles de disponibilité des ressources des nœuds du domaine. Il reçoit ces informations du *module de surveillance*. Cette base de données est utilisée par le module d'allocation des ressources afin de déterminer la stratégie la plus adéquate pour le déploiement du conteneur de l'application en fonction du comportement des nœuds du *fog*.

3.2.2 Le nœud du *fog*

Les nœuds du *fog* sont les dispositifs ou les infrastructures dotés de ressources de calculs, de stockage et de communication et se trouvant à la périphérie du réseau. Ces dispositifs peuvent appartenir à des entités distinctes. Leurs ressources peuvent être totalement ou partiellement dédiées à l'infrastructure du *fog*. Plusieurs types de nœuds hétérogènes ont été cités dans la littérature (Cisco, 2015; Marín-Tordera *et al.*, 2017) tels que les routeurs, commutateurs, points d'accès, caméras de vidéo-surveillance, les serveurs, les petits centres de données, les téléphones intelligents, les ordinateurs personnels, les voitures, etc.

Les nœuds du *fog* sont responsables du partage de leurs ressources, d'entreprendre les tâches qui leur sont assignées, de surveiller et de signaler les ressources de calcul et de communication disponibles au contrôleur de domaine, ainsi que de surveiller leur déplacement. Chaque nœud est géré par un unique contrôleur. Nous élucidons dans cette sous-section l'architecture d'un nœud du *fog* à travers la Figure 3.3.

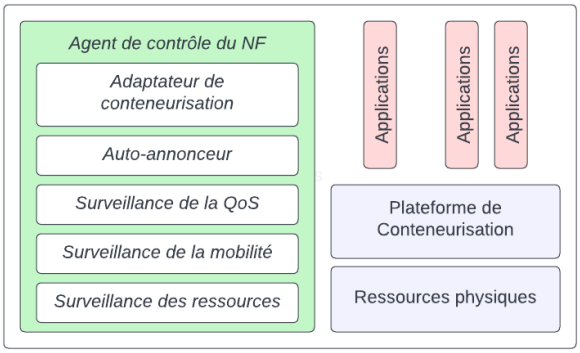


FIGURE 3.3 Architecture d'un nœud du *fog*

— *Ressources physiques* : Le module de ressources représente la base de l'architecture des nœuds du *fog*. Il représente les ressources physiques des nœuds incluant les ressources de calcul, de mémoire, de stockage et des ressources de communication.

— *Gestionnaire de Conteneurisation* : Chaque nœud doit inclure une plateforme de conteneurisation telle que Docker, LXC, etc., pour gérer localement ses conteneurs. Ce module assurera d'une part l'isolation des ressources des différentes applications pour les différents utilisateurs, mais aussi l'isolation de ces applications par rapport aux fonctions de base des nœuds.

— *Moniteur de ressources* : Les nœuds du fog sont chargés de surveiller et de signaler leurs ressources et les conditions de communication aux contrôleurs de domaine. Ce composant surveillera les ressources localement. Cependant, les contrôleurs de domaine maintiennent l'état des ressources disponibles et les conditions de communication de tous les nœuds du domaine. Grâce au cadre centralisé du contrôleur du *fog*, les ressources, la mobilité et la sécurité des nœuds peuvent être gérées efficacement pour atteindre un niveau élevé de qualité de service.

— *Moniteur de mobilité* : Les nœuds du fog sont chargés de surveiller et de signaler leurs modèles de mobilité permettant ainsi aux contrôleur du *fog*, de gérer efficacement les ressources et atteindre un niveau élevé de qualité de service.

— *L'adaptateur de conteneurisation* : Ce module sert à abstraire la solution de virtualisation sous-jacente et à interagir avec les différents modules du contrôleurs.

— *Moniteur de la QoS* : Il fournit des modules de surveillance et d'analyse pour détecter les dégradations de la QoS des applications déployées. Il interagit avec le contrôleur qui prendra des décisions de migration des conteneurs des applications si nécessaire.

— *Auto-annonceur* : Lorsqu'il est connecté à un réseau, le nœud du *fog* peut s'annoncer, en indiquant s'il agit en tant que nœud offrant ses ressources pour le transfert de tâches ou en tant que contrôleur de domaine. Il peut être ajouté à un domaine s'il reçoit des réponses du contrôleur lui demandant de le rejoindre. Des mécanismes d'incitation et de récompense peuvent être appliqués par les contrôleurs pour encourager les dispositifs inter-connectés à rejoindre l'infrastructure de *fog* et pour partager leurs ressources inutilisées.

3.3 Modélisation du système

Dans notre modèle, nous notons T le nombre considéré d'intervalles de temps dans le problème. Soit $\mathcal{F}$ l'ensemble des nœuds du *fog*. La matrice $\mathbf{F}_L$ de taille $|\mathcal{F}| \times T$ représente la localisation géographique du nœud à travers le temps. Chaque nœud $f \in \mathcal{F}$ a une certaine capacité de ressources, représentée par la matrice $\mathbf{F}_T$ de taille $|\mathcal{F}| \times |\mathcal{R}| \times T$, où $\mathcal{R}$ représente l'ensemble des types de ressources considérées. Il est à noter que ces ressources ne sont pas entièrement dédiées à l'infrastructure du *fog*. En effet, les ressources sont en priorité allouées aux fonctions de base de

chaque nœud, tandis que les ressources restantes sont allouées aux applications du *fog*. Cependant, la charge des fonctions de base peut varier dans le temps, ce qui entraîne une variation de la disponibilité des ressources pour le système du *fog*. On dénote $\mathbf{F}_B$ la matrice de taille $|\mathcal{F}| \times |\mathcal{R}| \times T$ représentant la quantité des ressources allouées aux fonctions de base à travers le temps où $|\mathcal{R}|$ ressources sont considérées.

Soit $\mathcal{A}$ l'ensemble des applications. Chaque application $a \in \mathcal{A}$ nécessite une quantité $A_R(a, r)$ de ressources pour chaque type r afin de traiter la charge $A_L(a)$.

Finalement, nous appelons $\mathcal{U}$ l'ensemble des utilisateurs. Les utilisateurs envoient des requêtes aux applications déployées dans des conteneurs, nous désignons par $U_A(u, a, t)$ les requêtes envoyées par l'utilisateur u pour l'application a à l'instant t, et $U_L(u, t)$ l'emplacement de l'utilisateur u à l'instant t. Étant donné que les requêtes des utilisateurs peuvent être gérées par plusieurs nœuds du *fog*, nous définissons par $\mathbf{U}_F$ la matrice binaire représentant si un utilisateur u est servi par le nœud f pour l'application a à l'intervalle de temps t. De plus, soit $\mathbf{U}_R(u, a, f, t)$ le nombre de requêtes de l'utilisateur u pour l'application a gérées par le nœud fog f à l'instant t. Finalement, nous définissons par $\mathbf{M}(u, a, f, t)$ l'indicateur qu'un nouveau déploiement de conteneur ou d'une migration dans le nœud fog f, afin de répondre aux requêtes de l'utilisateur u pour l'application a dans l'instant t. Les tableaux 3.1 et 3.2 résument les paramètres et variables définis.

Le placement des conteneurs dans les nœuds du fog a un impact direct sur le délai de réponse aux demandes des utilisateurs. Les nœuds du fog peuvent être désactivés et une grande partie de la consommation d'énergie peut être réduite grâce à la consolidation des conteneurs (Zhang *et al.*, 2013). Cependant, si un grand nombre de conteneurs sont fusionnés dans un nœud, le risque de sur-utilisation des ressources augmentera considérablement. De plus, les nœuds

du fog sont destinés à traiter en priorité leurs fonctions de base. Les ressources disponibles restantes sont allouées aux conteneurs du fog. De plus, un pic de demandes des ressources demandées pour les tâches des fonctions principales peut se produire généralement d'une manière inattendue. Par conséquent, les ressources de ces nœuds doivent être mises à disposition pour les fonctions de base autant que possible. Il existe donc un compromis entre le délai de traitement et la consommation d'énergie.

Compte tenu de ce modèle de système, notre objectif est de déterminer le placement optimal des conteneurs et l'approvisionnement des tâches qui maximise le nombre de requêtes satisfaites par rapport à un seuil de délai de réponse, sous des contraintes de déploiement d'un nombre minimal de conteneurs et en respectant les ressources disponibles des nœuds du fog. Dans ce qui suit, nous définissons les expressions du délai de service et d'utilisation des ressources pour la formulation du problème d'optimisation.

3.3.1 Modélisation mathématique des délais de réponse et de l'utilisation des ressources

- Délai de réponse

Le délai de réponse total pour les requêtes $U_A(u, a, t)$ d'un utilisateur u pour une application a déployé dans le nœud f à l'intervalle t est dénoté d_r et est calculé dans l'équation (3.1). Ce délai est composé du : (1) délai de communication entre l'utilisateur et le nœud du fog, (2) délai de déploiement du conteneur et d'installation de l'application et (3) délai de traitement de la requête au sein du nœud du fog.

Table 3.1 Les entrées du problème

Entrée	**Définition**
T	Nombre total d'intervalles.
Q	Seuil maximal du délai de réponse.
$\mathcal{A}$	Ensemble d'applications, où $\mathcal{A} = \{a_1, \ldots, a_{\lvert\mathcal{A}\rvert}\}$.
$\mathcal{F}$	Ensemble des nœuds du fog, où $\mathcal{F} = \{f_1, \ldots, f_{\lvert\mathcal{F}\rvert}\}$.
$\mathcal{R}$	Ensemble des type de ressources, où $\mathcal{R} = \{r_1, \ldots, r_{\lvert\mathcal{R}\rvert}\}$.
$\mathcal{U}$	Ensemble des utilisateurs , où $\mathcal{U} = \{u_1, \ldots, u_{\lvert\mathcal{U}\rvert}\}$.
$\mathbf{a}_L$	Vecteur de taille $\lvert\mathcal{A}\rvert$ représentant la charge d'utilisateurs supportée par une application, où $\mathbf{a}_L(a)$ est le nombre de tâches supportées par l'application a requérant $\mathbf{A}_R(a,r)$ ressources , $\forall 1 \leq r \leq \lvert\mathcal{R}\rvert$.
$\mathbf{A}_R$	Matrice de taille $\lvert\mathcal{A}\rvert \times \lvert\mathcal{R}\rvert$ représentant les ressources exigées par une application, où $\mathbf{A}_R(a,r)$ est l'exigence pour la ressource r de l'application a pour satisfaire la charge $\mathbf{a}_L(a)$.
$\mathbf{F}_L$	Matrice de taille $\lvert\mathcal{F}\rvert \times T$ représentant les localisations des nœuds du fog, où l'élément $\mathbf{F}_L(f,t)$ représente la localisation du nœud f à l'intervalle t dans un plan Cartésien.
$\mathbf{F}_T$	Matrice de taille $\lvert\mathcal{F}\rvert \times \lvert\mathcal{R}\rvert$ représentant la capacité maximale des ressources des nœuds du fog , où $\mathbf{F}_T(f,r)$ est la capacité du nœud f pour le type de ressource r .
$\mathbf{F}_B$	Matrice de taille $\lvert\mathcal{F}\rvert \times \lvert\mathcal{R}\rvert \times T$ représentant la quantité de ressources aalouée aux fonctions de base d'un nœud du fog, où $\mathbf{F}_B(f,r,t)$ est la quantité de ressource du type r allouée au fonctions de base du nœud f à l'intervalle t.
$\mathbf{u}_L$	Matrice de taille $\lvert\mathcal{U}\rvert \times T$ représentant la localisation des utilisateurs, où $\mathbf{u}_L(u,t)$ représente la localisation de l'utilisateur u à l'intervalle t
$\mathbf{U}_A$	Matrice de taille $\lvert\mathcal{U}\rvert \times \lvert\mathcal{A}\rvert \times T$ représentant la charge des utilisateurs pour les applications aux T intervalles, où $\mathbf{U}_A(u,a,t)$ est le nombre de requêtes de l'utilisateur u pour l'application a à l'intervalle t.

$$\begin{aligned} d_r(u,a,f,t) &= d_{com}(u,a,f,t) + d_{dep}(u,a,f,t) \\ &+ d_{cmp}(u,a,f,t), \end{aligned} \tag{3.1}$$

TABLE 3.2 Les variables du problème

Variable	Définition
$\mathbf{U}_F$	Matrice binaire de taille $\|\mathcal{U}\| \times \|\mathcal{A}\| \times \|\mathcal{F}\| \times T$ représentant l'association entre les nœuds du fog et les utilisateurs , oz $\mathbf{U}_F(u, a, f, t)$ indique si l'utilisateur u est servi par le nœud f pour l'application a à l'intervalle t ou non.
$\mathbf{U}_R$	Matrice de taille $\|\mathcal{U}\| \times \|\mathcal{A}\| \times \|\mathcal{F}\| \times T$ représentant le nombre de requêtes/tâches que les nœuds du fog servent aux utilisateurs, où $\mathbf{U}_R(u, a, f, t)$ est le nombre de requêtes/tâches servis à l'utilisateur u, pour l'application a par le nœud f à l'intervalle t.
$\mathbf{S}$	Matrice binaire de taille $\|\mathcal{U}\| \times \|\mathcal{A}\| \times \|\mathcal{F}\| \times T$ indiquant si les délais de service des requêtes des utilisateurs sont inférieurs au seuil Q, où $\mathbf{S}(u, a, f, t)$ indique si le délai de service de l'utilisateur u à l'intervalle t pour l'application a servi par le nœud f est inférieur à Q.
$\mathbf{M}$	Matrice binaire de taille $\|\mathcal{U}\| \times \|\mathcal{A}\| \times \|\mathcal{F}\| \times T$ indiquant la necessité du déploiement ou de la migration d'un conteneur à un nœud du fog, où $\mathbf{M}(u, a, f, t)$ indique si un deploiement ou une migration d'un conteneur de l'application a au nœud f à l'intervalle t pour répondre aux requêtes de l'utilisateur u.
$\mathbf{N}$	Matrice de taille $\|\mathcal{A}\| \times \|\mathcal{F}\| \times T$ représentant le nombre de conteneurs dans les nœuds du fog à chaque intervalle, où $\mathbf{N}(a, f, t)$ est le nombre de conteneurs pour l'application a déployé dans le nœud f à l'intervalle t.

où d_{com}, d_{dep} et d_{cmp} sont les délais de communication, de déploiement de l'application et d'exécution des requêtes respectivement.

Le délai de communication d_{com} entre l'utilisateur u et le nœud f est calculé dans l'équation 3.2.

$$d_{com}(u, a, f, t) = \mathbf{U}_F(u, a, f, t) \cdot (d_w(u, f, t) + d_w(f, u, t)), \tag{3.2}$$

où $d_w(u, f, t)$ (respectivement $d_w(f, u, t)$) est le délai de transmission entre l'utilisateur u et le nœud f (respectivement du nœud f à l'utilisateur u) à l'intervalle t, exprimé par l'équation 3.3 (Rappaport *et al.*, 1996). Pour le modèle

du canal, nous adoptons le modèle log-distance avec l'exposant d'atténuation sur la distance $\beta \geq 2$ (Gupta et Kumar, 2000). Dans ce modèle, la densité spectrale de la puissance du bruit est une constante notée N_0. Nous considérons B la bande passante du canal de transmission, P_u la puissance de transmission de l'utilisateur.

$$d_w(u, f, t) = \frac{L_u}{B \cdot \log_2 \left(1 + \frac{P_u \cdot \|\mathbf{u}_L(u) - \mathbf{F}_L(f,t)\|^{-\beta}}{N_0}\right)}, \tag{3.3}$$

où L_u la taille du paquet transmis par l'utilisateur u et $\|.\|$ est la norme Euclidienne .

Le délai de déploiement de l'application d_{dep} doit inclure le délai de déploiement du conteneur, le délai de téléchargement des paquets de l'application et le délai d'installation de l'application dans le conteneur . Ce délai n'est requis que lors d'un déploiement initial ou de migration d'un conteneur. Ce délai est calculé dans l'équation 3.4.

$$d_{dep}(u, a, f, t) = \mathbf{M}(u, a, f, t) \cdot d_c(a), \tag{3.4}$$

où $d_c(a)$ est la somme des intervalles requis pour le déploiement d'un conteneur, son téléchargement et l'installation des paquets l'application a dans le conteneur.

Finalement, le délai d'exécution d_{cmp}, est relié aux capacités d'exécution d'un nœud du fog pour exécuter l'application a, défini par $d_e(a, f)$, et à la charge des requêtes des utilisateurs pris en charge par ce nœud. Le délai d'exécution est exprimé dans l'équation 3.5.

$$d_{cmp}(u, a, f, t) = \mathbf{U}_R(u, a, f, t) \cdot d_e(a, f). \tag{3.5}$$

Utilisation des ressources

Afin de ne pas compromettre l'extensibilité des fonctions de base, nous plaçons les conteneurs dans les nœuds de manière à minimiser les ressources allouées dans chaque nœud. Ceci permettra de laisser assez de ressources disponibles pour l'extensibilité des fonctions de base. Le taux d'utilisation d'une ressource r dans un nœud f est exprimé dans l'équation 3.6.

$$\gamma(f,r,t) = \frac{\sum_{a=1}^{|\mathcal{A}|} \mathbf{N}(a,f,t) \cdot \mathbf{A}_R(a,r)}{\mathbf{F}_T(f,r) - \mathbf{F}_B(f,r,t)}. \tag{3.6}$$

3.4 Formulation du problème

L'objectif de notre problème est de maximiser le nombre de requêtes satisfaites i.e. la réponse est reçu dans un délai de réponse inférieur à un seuil prédéterminé Q.

Notre modèle est soumis aux contraintes suivantes :

$$\begin{aligned} Q - d_r(u,a,f,t) \quad &\geq \quad G \cdot (\mathbf{S}(u,a,f,t) - 1), \\ &\qquad \forall u \in \mathcal{U}, a \in \mathcal{A}, f \in \mathcal{F}, 1 \leq t \leq T, \end{aligned} \tag{3.7}$$

où G est un très grand nombre positif sélectionné aléatoirement, et $\mathbf{S}(u,a,f,t)$ est une variable binaire indiquant si le délai de réponse $d_r(u,a,f,t)$ est inférieur au seuil Q ou non. En effet, la contrainte (3.7) assure que $\mathbf{S}(u,a,f,t)$ est égal à 0 si le délai de réponse $d_r(u,a,f,t)$ est supérieur au seuil Q. Nous assurons aussi dans la contrainte (3.8) que la requête de l'utilisateur u n'est satisfaite par le nœud f que si l'utilisateur u est associé au nœud f.

$$\begin{aligned} \mathbf{S}(u,a,f,t) &\leq \mathbf{U}_F(u,a,f,t), \\ &\forall u \in \mathcal{U}, a \in \mathcal{A}, f \in \mathcal{F}, 1 \leq t \leq T. \end{aligned} \tag{3.8}$$

Le modèle garanti que la somme totale des ressources allouées à tous les conteneurs déployés dans un unique nœud ne dépasse pas les capacités des ressources de ce nœud. Ainsi, la contrainte (3.9) assure que le taux d'utilisation, défini dans l'équation 3.6, de chaque ressource dans chaque nœud du *fog* est inférieur à 1.

$$\gamma(f,r,t) \leq 1, \ \forall f \in \mathcal{F}, r \in \mathcal{R}, 1 \leq t \leq T. \tag{3.9}$$

De plus, la contrainte (3.10) assure que pour chaque utilisateur, la somme des requêtes servies par tous les nœuds du *fog*, n'excède pas la charge des requêtes que cet utilisateur a demandé.

$$\begin{aligned} \sum_{f=1}^{|\mathcal{F}|} \mathbf{U}_R(u,a,f,t) &\leq \mathbf{U}_A(u,a,t), \\ &\forall u \in \mathcal{U}, a \in \mathcal{A}, 1 \leq t \leq T. \end{aligned} \tag{3.10}$$

D'une autre part, la contrainte (3.11) assure que l'utilisateur u n'est associé au nœud f (i.e. $\mathbf{U}_F(u,a,f,t) = 1$) que si une partie de ses requêtes est servie par ce nœud.

$$\begin{aligned} \mathbf{U}_F(u,a,f,t) &\leq \mathbf{U}_R(u,a,f,t), \\ &\forall u \in \mathcal{U}, a \in \mathcal{A}, f \in \mathcal{F}, 1 \leq t \leq T. \end{aligned} \tag{3.11}$$

Dans le but de mieux utiliser les ressources des nœuds du fog, nous déployons un nombre minimal de conteneurs afin de satisfaire les requêtes des utilisateurs. Ceci est appuyé par les contraintes (3.12) et (3.14) où le nombre des conteneurs déployés dans tous les nœuds est proportionnel au nombre des requêtes qu'ils servent.

$$\begin{aligned} \sum_{u=1}^{|\mathcal{U}|} \mathbf{U}_R(u,a,f,t) &\leq \mathbf{a}_L(a) \cdot \mathbf{N}(a,f,t), && (3.12) \\ \sum_{u=1}^{|\mathcal{U}|} \mathbf{U}_R(u,a,f,t) &\geq \mathbf{a}_L(a) \cdot (\mathbf{N}(a,f,t)-1), && (3.13) \\ & \forall a \in \mathcal{A}, f \in \mathcal{F},\ 1 \leq t \leq T. \end{aligned}$$

Au premier intervalle (i.e. $t = 1$) si un utilisateur u est servi par le nœud f, un déploiement initial d'un conteneur est nécessaire. La contrainte (3.15) assure ce déploiement initial.

$$\begin{aligned} \mathbf{M}(u,a,f,1) &\geq \mathbf{U}_F(u,a,f,1), && (3.14) \\ & \forall u \in \mathcal{U}, a \in \mathcal{A}, f \in \mathcal{F}. \end{aligned}$$

Finalement, la somme des requêtes des utilisateurs requérant de nouveaux déploiements de conteneurs ou leurs migrations dans un nœud f est proportionnel à la différence de charge de ce nœud f à l'instant t et celle qui aurait pu être gérée avec les conteneurs déployés à l'instant $t-1$. Ceci est assuré avec la contrainte (3.15).

$$\begin{aligned} \sum_{u=1}^{|\mathcal{U}|} \mathbf{M}(u,a,f,t) \cdot \mathbf{U}_R(u,a,f,t) &\geq \sum_{u=1}^{|\mathcal{U}|} \mathbf{U}_A(u,a,f,t) \\ &- \mathbf{N}(a,f,t-1) \cdot \mathbf{a}_L(a), \\ \forall a \in \mathcal{A}, f \in \mathcal{F},\ 2 \leq t \leq T. & \qquad (3.15) \end{aligned}$$

La partie gauche de l'équation dans la contrainte (3.15) peut être linéarisée selon les règles développées dans (Coelho, 2013).

En raison du comportement dynamique et de la mobilité des nœuds du *fog*, le délai de réponse des requêtes des utilisateurs est dégradé. Par conséquent, nous visons à minimiser ce délai dans un environnement de fog computing dynamique. Nous allouons des ressources aux applications dans les nœuds du fog, tout en tenant compte de leur emplacement et de leur délai de communication. D'autre part, comme les fonctions de base des nœuds physiques ne doivent pas être affectées par les conteneurs du fog, nous visons à maximiser autant que possible les ressources disponibles à ces nœuds afin de ne pas compromettre leurs extensibilité. Notre problème consiste à trouver la meilleure stratégie de placement de conteneurs dans le but de maximiser du nombre de requêtes avec un délai de réponse inférieur à un seuil fixé Q. La fonction objectif est exprimée dans l'équation (3.16)

$$\max_{\mathbf{U}_R} \sum_{t=1}^{T} \sum_{u=1}^{|\mathcal{U}|} \sum_{a=1}^{|\mathcal{A}|} \sum_{f=1}^{|\mathcal{F}|} \frac{\mathbf{S}(u,a,f,t) \cdot \mathbf{U}_R(u,a,f,t)}{\mathbf{U}_A(u,a,t)} \tag{3.16}$$

3.5 Étude de la complexité du problème

Le problème modélisé dans (3.7)-(3.16) est NP-Difficile. Dans le but de le prouver, nous étudions le cas simple de ce problème considérant un seul intervalle de temps, visant à placer les conteneurs dans les nœuds du fog dans le but de maximiser le nombre de requêtes d'utilisateurs satisfaites. Nous simplifions encore plus le problème en considérant que les requêtes des utilisateurs sont déjà assignées aux conteneurs qui vont exécuter leurs tâches. Ces conteneurs peuvent avoir différents besoins en ressources distinctes. Ce problème particulier peut être facilement assimilé au problème bien connu d'affectation généralisé (Generalized Assignment Problem) (Ross et Soland, 1975).

Dans ce problème, l'objectif est de trouver une affectation de n tâches avec des besoins de ressources différents à m agents avec un profit maximal. Il faut que chaque tâche soit affectée précisément à un agent, sous réserve de la restriction de capacité de ressources des agents. De plus, chaque tâche a un profit différent, selon l'agent affecté. Logiquement, les agents peuvent être assimilés aux nœuds du fog et les conteneurs à des tâches, où le profit de la tâche est le nombre total de requêtes satisfaites exécutées dans le conteneur. Puisque GAP est connu pour être NP-Difficile, alors, par restriction, notre version simplifiée du problème est également NP-Difficile (Garey et Johnson, 2002).

3.6 Conclusion

Dans ce chapitre, nous avons présenté l'architecture de l'infrastructure considéré. Puis nous avons modélisé notre système de fog dynamique et formulé le problème conjoint de placement de conteneurs et d'approvisionnement de tâches est formulé à l'aide de la programmation en nombre entier. Ensuite, nous avons prouvé la NP-Difficulté du problème étudié. En outre, le problème (3.7)-(P1) a été modélisé en utilisant le langage AMPL (AMPL, 2016). Le problème a ensuite été résolu et la solution optimale obtenue à l'aide de l'outil IBM CPLEX Optimizer (CPLEX, 2016b).

CHAPITRE IV

MÉTHODES OFFLINES POUR L'ALLOCATION DE RESSOURCES

4.1 Introduction

Étant donné la NP-Difficulté du problème (3.7)-(3.16), sa résolution pour l'obtention de la solution optimale en utilisant CPLEX est caractérisée par une haute complexité temporelle. Pour cette raison, nous proposons dans cette section deux algorithmes sous-optimaux à faible complexité temporelle pour la résolution du problème étudié.

Ainsi, nous proposons d'abord une méta-heuristique basée sur l'Optimisation à Essaims Particulaires (OEP). OEP permet de réduire le temps d'exécution en comparaison avec l'optimiseur CPLEX . OEP est aussi sélectionné parmi d'autres algorithmes parce qu'il ne nécessite pas d'opérations de croisement, de décodage ou d'encodage comme pour les algorithmes génétiques et contrairement à d'autres algorithmes bio-inspirés tels que l'optimisation des colonies de fourmis (Kennedy et Eberhart, 1995; Yu *et al.*, 2004). La méta-heuristique basée sur OEP permet aussi de maximiser le nombre de requêtes satisfaites dans un délai de réponse inférieur à un seuil Q tout en respectant les contraintes liées à la capacité des serveurs et aux nombres de conteneurs requis pour répondre aux requêtes des utilisateurs.

Par ailleurs, le nombre considérable de nœuds du *fog*, ainsi que le comportement

dynamique de ces nœuds, augmente la complexité et la dimension du problème. Par conséquent, nous proposons aussi dans ce chapitre un algorithme glouton, qui se base sur quelques données liées aux comportements des utilisateurs finaux et des nœuds du *fog*. Cet algorithme est capable d'atteindre des performances intéressantes avec une très faible complexité.

4.2 Algorithme basé sur l'Optimisation à Essaims Particulaires

4.2.1 L'Optimisation à Essaims Particulaires

Les algorithmes évolutionnaires, élaborés au cours des années 1950, forment une famille d'algorithmes de recherche inspirés de l'évolution biologique des espèces. L'idée ici est de s'inspirer de la théorie darwiniste de sélection naturelle pour résoudre des problèmes d'optimisation. On peut distinguer trois grandes classes d'algorithmes évolutionnaires : les algorithmes génétiques (Golberg, 1989), les stratégies d'évolution (Schwefel, 1981) et la programmation évolutive (Fogel, 2006).

L'Optimisation à Essaims Particulaires (OEP), ou Particle Swarm Optimization (PSO) en anglais, est un algorithme évolutionnaire qui utilise une population de solutions candidates pour développer une solution optimale au problème. Cet algorithme a été proposé par Russel Eberhart (ingénieur en électricité) et James Kennedy (socio-psychologue) en 1995 (Kennedy et Eberhart, 1995).

OEP est une approche stochastique basée sur la population qui vise à résoudre des problèmes d'optimisation difficiles à variables continues et discrètes. Cette approche est inspirée du comportement social du flocage des oiseaux ou au déplacement des poissons en bancs. L'idée directrice de cette méthode est de simuler le comportement collectif des oiseaux à l'intérieur d'une nuée : leur capacité à voler de façon synchrone et leur aptitude à changer brusquement de

direction, tout en restant en une formation optimale.

OEP est considéré comme ayant un concept simple, une mise en œuvre facile et une convergence rapide (Yu *et al.*, 2004). C'est pour ces raisons que cet algorithme a gagné beaucoup d'attention et une large application dans différents domaines. OEP a également été reconnu pour être robuste dans la résolution des problèmes se caractérisant par la non-linéarité, la non-différentiabilité et une haute dimensionnalité (Akjiratikarl *et al.*, 2007).

Principe général

L'essaim de particules correspond à une population d'agents simples, appelés particules. Chaque particule est considérée comme une solution du problème, où elle possède une position (le vecteur solution) et une vitesse. De plus, chaque particule possède une mémoire lui permettant de se souvenir de sa meilleure performance (en position et en valeur) et de la meilleure performance atteinte par les particules « voisines ». Un essaim de particules, qui sont des solutions potentielles au problème d'optimisation, « survole » l'espace de recherche, à la recherche de l'optimum global.

Le déplacement d'une particule est influencé par les trois composantes suivantes :

- Une composante physique : la particule tend à suivre sa direction courante de déplacement ;
- Une composante cognitive : la particule tend à se diriger vers le meilleur site par lequel elle est déjà passée ;
- Une composante sociale : la particule tend à se fier à l'expérience de ses congénères et, ainsi, à se diriger vers le meilleur site déjà atteint par ses voisins.

La Figure 4.2.1 illustre la stratégie de déplacement d'une particule.

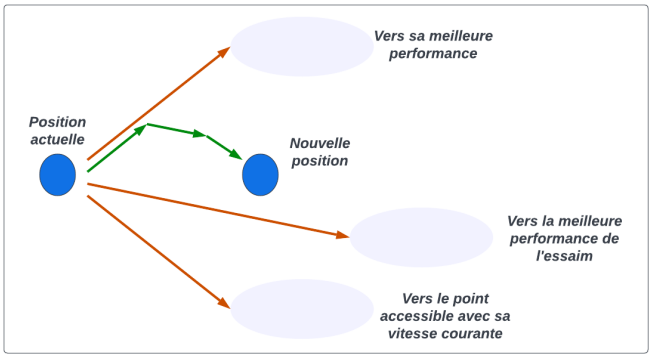


FIGURE 4.1 Déplacement d'une particule

Formalisation

Dans un espace de recherche de dimension D, la particule i de l'essaim est modélisée par son vecteur position $\overrightarrow{X_i} = \{x_{i_1}, x_{i_2}, \cdots, x_{i_D}\}$ et par son vecteur vitesse $\overrightarrow{V_i} = \{v_{i_1}, v_{i_2}, \cdots, v_{i_D}\}$. Cette particule garde en mémoire la meilleure position par laquelle elle est déjà passée, que l'on note $\overrightarrow{pBest_i} = \{pBest_{i_1}, pBest_{i_2}, \cdots, pBest_{i_D}\}$. La meilleure position atteinte par toutes les particules de l'essaim est notée $\overrightarrow{gBest_i} = \{gBest_{i_1}, gBest_{i_2}, \cdots, gBest_{i_D}\}$

L'algorithme OEP (Clerc et Kennedy, 2002) commence par la génération de positions aléatoires des particules dans l'espace de recherche. Les vitesses sont généralement initialisées à l'intérieur de l'espace de recherche, mais peuvent également être initialisées à zéro ou à de petites valeurs aléatoires pour empêcher les particules de quitter l'espace de recherche au cours des premières itérations. À chaque itération, la particule est mise à jour suivant les deux meilleures solutions $pBest$ et $gBest$.

Au cours de la boucle principale de l'algorithme, à chaque itération t, la vitesse (V_i^{t+1}) et la position (X_i^{t+1}) de chaque particule i sont mises à jour selon les règles

(4.1) et (4.2) respectivement. Cette boucle prend fin quand un critère d'arrêt est atteint (nombre d'itérations maximal par exemple).

$$V_i^{t+1} = w \cdot V_i^t + \phi_1 \cdot U_1^t \cdot (gBest^t - X_i^t) + \phi_2 \cdot U_2^t \cdot (pBest_i^t - X_i^t) \tag{4.1}$$

$$X_i^{t+1} = X_i^t + V_i^t \tag{4.2}$$

Dans l'Équation (4.1), w est un paramètre appelé masse d'inertie, ϕ_1 et ϕ_2 sont deux paramètres appelés coefficients d'accélération. Si les valeurs de w, ϕ_1, ϕ_2 sont bien choisis, il est garanti que les vitesses des particules n'augmenteront pas à l'infini (Clerc et Kennedy, 2002). U_1^t et U_2^t sont deux matrices diagonales $n \times n$, où les valeurs de la diagonale principale sont des nombres aléatoires répartis uniformément dans l'intervalle [0, 1), et n est le nombre de particules. À chaque itération t, ces matrices sont régénérées.

Pour évaluer une particule, on utilise une fonction de *fitness* $f(X_i^t)$. La fonction de *fitness* est la fonction à optimiser dans le problème. $pBest_i^t$ est la meilleure position trouvée par la particule i jusqu'à l'itération t. Ainsi, $\forall k \in [0, t] \quad f(pBest_i^t)$ est meilleure que $f(X_i^k)$. Et, $gBest^t$ est la meilleure position trouvée jusqu'à l'itération t par toutes les particules.

Une fois le déplacement des particules effectué, les nouvelles positions sont évaluées et les deux vecteurs $pBest_i^t$ et $gBest^t$ sont mis à jour , à l'itération $t+1$, suivant les deux équations 4.3 (dans le cas d'une minimisation) et 4.4 (dans une version globale de OEP), respectivement

$$pBest_i^{t+1} = \begin{cases} pBest_i^t & \text{si } f(X_i^t) \geq pBest_i^t \\ X_i^{t+1} & \text{sinon.} \end{cases} \tag{4.3}$$

$$gBest^{t+1} = arg \min_{pBest_i} f(pBest_i^{t+1}), 1 \leq i \leq n \tag{4.4}$$

Une fois la condition d'arrêt est atteinte, la solution au problème est représentée par le $gBest^t$.

Comme la majorité des méta-heuristiques, l'OEP nécessite le réglage au préalable de plusieurs paramètres de contrôle en fonction du problème considéré. Les performances de l'OEP présentent une forte corrélation avec le réglage de ces paramètres. Il est donc indispensable pour un concepteur d'étudier l'influence de chaque paramètre sur le comportement de l'algorithme afin de déterminer le jeu de paramètres optimal (Smairi, 2013).

4.2.2 Algorithme proposé

La particule

La méta-heuristique basée sur le OEP donne lieu à une stratégie d'allocation des ressources qui devrait atteindre des performances quasi optimales avec un nombre limité d'itérations. En effet, les particules se déplacent dans l'espace de recherche du problème d'optimisation, où la position d'une particule représente une solution candidate. Chaque particule explore une meilleure position dans l'espace de recherche en modifiant sa vitesse. Pour adapter cette méta-heuristique à notre problème, nous utilisons les entrées et les variables décrites dans les Tables 3.1 et 3.2 respectivement. La particule i est représentée par une matrice $\mathbf{X}_i$ de taille $|\mathcal{U}| \times |\mathcal{A}| \times |\mathcal{F}| \times T$. La matrice $\mathbf{X}_i$ représente en effet, l'association entre les utilisateurs et les nœuds du fog, où les conteneurs des applications seront déployés à chaque intervalle, d'une manière similaire à $\mathbf{U}_R$. Nous définissons $\mathcal{N} = \{1, \ldots, n\}$ l'ensemble des particules.

Calcul de la vitesse et de la position

Pour le calcul de la position et de la vitesse de chaque particule, nous devons mettre à jour tous les éléments de la matrice X_i. Ainsi, $pBest_i^t$ et $gBest_i^t$ sont aussi des matrices de dimension $|\mathcal{U}| \times |\mathcal{A}| \times |\mathcal{F}| \times T$. Par ailleurs, nous calculons la vitesse V_i^{t+1} suivant la formule 4.1 et la position X_i^{t+1} suivant la formule 4.2.

Validation de la particule

Lorsqu'une particule viole l'une des contraintes imposées par notre modèle, nous la réparons en changeant sa position par une nouvelle qui respecte toutes les contraintes. Par exemple, si la capacité d'un nœud du fog est dépassée, nous supprimons au hasard certains conteneurs et nous redirigeons les requêtes des utilisateurs vers d'autres conteneurs. Nous pouvons aussi créer un nouveau conteneur dans un autre nœud, si il n'y a pas d'autre conteneur qui exécute cette application déjà déployé. De plus, nous vérifions que le nombre de requêtes répondues ne dépasse pas celui de requêtes demandées par les utilisateurs. Si nous ne parvenons pas à réparer une particule, nous la pénalisons en ajoutant une très grande valeur à sa fonction de fitness. Nous adoptons le nombre maximal d'itérations θ_{max} comme critère d'arrêt afin d'atteindre une complexité de calcul raisonnable, c'est-à-dire $\mathcal{O}(\theta_{max} \times |\mathcal{N}| \times |\mathcal{U}| \times |\mathcal{A}| \times |\mathcal{F}| \times T)$

Fonction de fitness

La fonction de *fitness* décrit les objectifs que nous voulons atteindre à l'issue de notre heuristique. Les objectifs de notre modèle est de placer les conteneurs dans les nœuds du fog et de diriger les requêtes des utilisateurs vers ces conteneurs, tout en maximisant le nombre de requêtes satisfaites dans un délai de réponse inférieur à un seuil Q. Cette fonction est décrite par l'équation 4.5. La particule $gBest^t$ sera la matrice X_i^t qui aura la valeur minimale de $f(X_i^t)$.

Algorithme 1 Algorithme basé sur OEP

Entrées : Nombre de particules n, Table 3.1 entrées, Condition d'arrêt

Sorties : Meilleure particule

$\mathcal{N} \leftarrow \emptyset$
$\theta \leftarrow 0$
pour each $i \in \mathcal{N}$ **faire**
    Initialiser la particule i ($\mathbf{X}_i^\theta$)
    $\mathcal{N} \leftarrow \mathcal{N} \cup \{i\}$
    $\text{pbest}_i \leftarrow \mathbf{X}_i^\theta$
fin pour
gbest$\leftarrow \mathbf{X}_{i_0}^\theta$, tel que Particule $i_0 = \arg\max_{i \in \mathcal{N}} \text{fitness}(\text{pbest}_i)$
tant que $\theta \leq \theta_{max}$ **faire**
    pour each $i \in \mathcal{N}$ **faire**
        Calculer la vélocité et la position de la particule en utilisant les équations (4.1)-(4.2)
        Vérifier que la particule i respecte toutes les contraintes
        si Une contrainte est violée et la particule i est réparable **alors**
            Réparer la particule i
        sinon
            si Une contrainte est violée et la particule i n'est pas réparable **alors**
                $\text{fitness}(\mathbf{X}_i^\theta) \leftarrow +\infty$
            fin si
        fin si
    fin pour
    pour each $i \in \mathcal{N}$ **faire**
        si $\text{fitness}(\mathbf{X}_i^\theta) \leq \text{fitness}(\text{pbest}_i)$ **alors**
            $\text{pbest}_i \leftarrow \mathbf{X}_i^\theta$
        fin si
    fin pour
    gbest $\leftarrow \mathbf{X}_{i_0}^\theta$ tel que Particule $i_0 = \arg\max_{i \in \mathcal{N}} \text{fitness}(\text{pbest}_i)$
    $\theta \leftarrow \theta + 1$
fin tant que
returner gbest

$$f(X) = \max_{\mathbf{U}_R} \sum_{t=1}^{T} \sum_{u=1}^{|\mathcal{U}|} \sum_{a=1}^{|\mathcal{A}|} \sum_{f=1}^{|\mathcal{F}|} \frac{\mathbf{S}(u,a,f,t) \cdot \mathbf{U}_R(u,a,f,t)}{\mathbf{U}_A(u,a,t)} \tag{4.5}$$

L'Algorithme 1 décrit notre solution.

4.3 Algorithme glouton

Nous proposons cette solution gloutonne afin de fournir un algorithme réalisable à faible complexité, adapté aux plateformes de calcul en temps réel des environnements de fog computing dynamiques. Cette approche est illustrée dans l'Algorithme 2.

Cet algorithme procède comme suit : à chaque intervalle, les utilisateurs sont triés en fonction du nombre décroissant de requêtes dans l'ensemble $\mathcal{U}'$. Pour chaque application a demandée par l'utilisateur $u \in \mathcal{U}'$, nous arrangeons selon une fonction de poids les nœuds du fog qui déploient déjà des conteneurs pour l'application que demande cet utilisateur dans un nouvel ensemble $\mathcal{F}'$. Cette fonction de poids combine deux métriques :

1. la distance moyenne a travers le temps entre l'emplacement des nœuds du fog et l'emplacement de l'utilisateur u ;
2. les ressources disponibles dans les nœuds du fog à travers le temps (*FogDeployingAppCont, Weight*).

Ensuite, nous allouons d'une manière itérative les requêtes des utilisateurs *u.req* aux conteneurs existants (*AddLoadToCont*), si le délai de réponse pour les requêtes de cet utilisateur est inférieur au seuil Q. Par conséquent, la "charge" représente le nombre de requêtes de l'utilisateur qui seront traitées par le conteneur dans le nœud $\mathcal{F}'(i)$.

Si les conteneurs existants ne peuvent pas gérer toutes les requêtes des utilisateurs,

Algorithme 2 Algorithme glouton

Entrées : Table 3.1 entrées

Sorties : $\mathbf{U}_R$

pour each $t = 1, \dots T$ **faire**
  $\mathcal{U}' \leftarrow$ Utilisateurs triés par ordre décroissant du nombre de requêtes
  pour each $u \in \mathcal{U}'$ **faire**
    pour each $a \in u.app$ **faire**
      $\mathcal{F}'$ =FogsDeployingAppCont(a)
      $\mathcal{F}'$ = Poids($\mathcal{F}'$,u.loc, $\mathbf{F}_L$, $\mathbf{F}_B$, $\mathbf{F}_T$, $\mathbf{U}_R$)
      $i \leftarrow 0$
      tant que u.req > 0 and $i \leq |\mathcal{F}'|$ **faire**
        charge = AddLoadToCont($\mathcal{F}'(i)$, u.req)
        si *load* > 0 **alors**
          u.req = u.req - charge
          $\mathbf{U}_R$(u.index, u.app, $\mathcal{F}'(i)$, t) = charge
        sinon
          $i \leftarrow i + 1$
        fin si
      fin tant que
      si u.req > 0 **alors**
        $\mathcal{F} \leftarrow$ Poids($\mathcal{F}$,u.loc, $\mathbf{F}_L$, $\mathbf{F}_B$, $\mathbf{F}_T$, $\mathbf{U}_R$)
        $i \leftarrow 0$
        tant que u.req > 0 and $i < |\mathcal{F}|$ **faire**
          charge = CreateContainers($\mathcal{F}(i)$, u.req)
          si *load* > 0 **alors**
            u.req = u.req - charge
            $\mathbf{U}_R$(u.index, u.app, $\mathcal{F}(i)$, t) = charge
          sinon
            $i \leftarrow i + 1$
          fin si
        fin tant que
      fin si
    fin pour
  fin pour
fin pour
retourner $\mathbf{U}_R$

ou si aucun conteneur de l'application demandée n'existe au sein de la plate-forme de fog computing, nous réorganisons les nœuds du fog en fonction des métriques utilisées auparavant pour l'arrangement des noeuds, puis nous créons suffisamment de conteneurs dans les nœuds du fog en respectant la contrainte de délai de service (*CreateContainers*). Ensuite, nous les utilisons pour satisfaire les requêtes des utilisateurs restantes. La complexité temporelle de l'algorithme est $\mathcal{O}(|\mathcal{U}| \times |\mathcal{A}| \times |\mathcal{F}| \times T)$.

4.4 Évaluation des performances

Dans cette section, nous évaluons les performances des algorithmes proposés dans ce chapitre en mettant en œuvre un système routier intelligent déployé dans un environnement de *fog computing* dynamique. Nous implémentons des scénarios qui nous permettent de comparer les performances de la solution optimale, de l'algorithme basé sur OEP, de l'algorithme glouton et d'un algorithme de l'état de l'art appelé Folo (Zhu *et al.*, 2019).

Folo est une plateforme déclenché par des événements qui utilise soit l'optimisation basée sur la programmation linéaire (LP) soit l'optimisation des essaims particulaires binaires (OEPB) pour résoudre les problèmes de provisionnement des tâches de calcul dans les nœuds du fog (Zhu *et al.*, 2019). Pour traiter la violation des contraintes, l'algorithme basé sur OEPB ne considère pas la réparation ou la pénalité des particules comme notre travail, mais il évite simplement de placer la particule dans une position qui ne représente pas une solution réalisable (en raison de la violation de contraintes). À chaque créneau horaire, Folo prend des décisions de provisionnement des tâches en tenant compte du placement des utilisateurs et des nœuds du fog à cet intervalle. Cependant, dans notre travail, les positions futures des nœuds sont également prises en compte dans le processus de provisionnement des tâches. Folo peut être adapté à notre contexte de mobilité,

en considérant les emplacements moyens des nœuds du fog au fil du temps, tandis que les migrations dans Folo sont effectuées via un processus basé sur les seuils. Dans notre travail, les migrations peuvent être effectuées à n'importe quel créneau horaire afin d'optimiser le nombre total de demandes d'utilisateurs satisfaits.

Pour obtenir la solution optimale, nous modélisons notre problème linéaire en nombres entiers en utilisant le langage AMPL (AMPL, 2016), puis nous le résolvons en utilisant l'outil IBM CPLEX Optimizer (CPLEX, 2016b). CPLEX utilise la recherche "branch-and-cut" lors de la résolution de modèles de programmation en nombres entiers (IP) (CPLEX, 2016a).

4.4.1 Description du cas d'utilisation

Le système routier intelligent s'appuie sur une combinaison de capteurs, notamment des caméras, des radars, des véhicules connectés, etc., pour détecter les conditions de la route et fournir une gamme de décisions pour la sécurité, la sûreté et le contrôle de la circulation. Plus précisément, les images, les vidéos et les données collectées par les caméras et les capteurs seront traitées pour en extraire des informations utiles telles que les types d'objets ainsi que leurs caractéristiques.

Ainsi, pour éviter les accidents des piétons, une application de reconnaissance d'objets est déployée pour reconnaître le piéton et le véhicule menaçant, à partir de vidéos en direct captées par les caméras de circulation. En outre, ces applications sensibles à la latence nécessitent des capacités de calcul que les capteurs et les caméras ne pourraient pas fournir. Par conséquent, nous considérons la plateforme de calcul basé sur une infrastructure de fog computing. Cette infrastructure est souvent recommandée pour les systèmes routiers intelligents. Elle permet en effet d'assurer une faible latence à ces applications étant donné la proximité des nœuds de traitement par rapport aux utilisateurs finaux. Cette infrastructure basée sur le fog est composée de trois couches. La première comprend les unités d'acquisition de

données, comme les capteurs, les caméras et les véhicules connectés. La couche du fog se compose d'unités de traitement routières (RSU), de véhicules connectés, des téléphones portables des piétons et des ordinateurs portables, où les applications peuvent être déployées dans des conteneurs. Enfin, la couche de cloud qui permet d'héberger des applications qui ne nécessitent pas d'exigences strictes pour la latence et permet aussi d'assurer le stockage des données à long terme.

Table 4.1 Types de nœuds du fog

Appareil	**Nombre de nœuds**	**Ressources disponibles**	
		vCPU	RAM
RSUs	3	256	64 GB
Voitures	18	32	8 GB
Téléphones intelligents	30	8	4 GB

Paramètres de l'environnement

Nous considérons un modèle de système couvrant une zone géographique de 500 m^2. Ce système est composé de 51 nœuds (sauf indication contraire) dédiés à la couche du fog et constitués de trois types d'appareils, comme décrit dans le tableau 4.1. Les ressources disponibles pour le traitement correspondent au CPU virtuel (vCPU) et à la mémoire (RAM). Un vCPU représente l'équivalent d'un cœur de CPU physique ou d'un *hyperthread* dans un processeur Intel avec la fonctionnalité de *Hyperthreading* activée (kubernetes, 2018)

Nous supposons également que les nœuds dans le réseau considéré transmettent des paquets de longueur $L = 1500$ bits sur un canal sans fil (IEEE Std., 2009), en utilisant un contrôle de puissance optimisé (aucune retransmission requise) et avec une bande passante $B = 20$ MHz. Nous supposons que les transmissions sans fil subissent un affaiblissement de coefficient $\beta = 3$ et une puissance de bruit

$N_0 = 0,02$ Watts.

Pour nos scénarios, nous considérons 20 caméras uniformément réparties dans la région géographique étudiée. Ces caméras transmettent des vidéos en temps réel aux nœuds du *fog* afin d'être traitées. Nous considérons deux applications pour le traitement des vidéos qui seront déployées dans des conteneurs dans les nœuds du *fog*.

L'application d'évitement des collisions **A1** détecte et suit les véhicules et les piétons afin de prévenir les accidents. Cette application analyse la dynamique des déplacements des véhicules et des piétons. Une agrégation de données spatio-temporelles permet de détecter une collision. Les besoins en ressources de cette application sont de 2 vCPU et de 2 Go de mémoire. Ces ressources correspondent aux exigences typiques qu'un conteneur a besoin pour analyser 15 images par seconde (fps). Le traitement de chaque image prend en moyenne 68 ms (Betke *et al.*, 2000). Le seuil de délai de réponse pour cette application est de Q=100ms (Araniti *et al.*, 2013). Ce délai devrait comprendre le temps de transmission de la vidéo de la caméra au nœud du fog, le temps traitement et le temps de transmission de la prédiction à la caméra. Un signal sonore fort peut avertir les véhicules et les piétons si une collision est prévue.

La deuxième application **A2** est dédiée à la reconnaissance des plaques d'immatriculation en temps réel utilisée pour des fins de sécurité (C. Arth et Bischof, 2007). Cette application est composée principalement de modules de détection et de reconnaissance de caractères. Pour traiter 20 images par seconde, le conteneur de l'application nécessite 1 vCPU et 1 Go de RAM. Le tableau 4.2 résume les exigences des applications **A1** et **A2**.

Table 4.2 Exigences des applications considérées

Application	Seuil du délai	Exigences en ressources		Images analysées
		vCPU	RAM	
A1	0.1 sec	2	2 Gb	15 fps
A2	0.1 sec	1	1 Gb	20 fps

Paramètres de l'OEP

Comme étudié dans la sous-section 4.2.1, les positions initiales des particules sont générées aléatoirement. Cependant, nous étudierons dans cette section une optimisation où les positions initiales des particules sont générées sur la base d'un écart aléatoire par rapport au résultat de l'algorithme glouton. En outre, dans les premières recherches effectuées par Eberhart et Shi (Eberhart et Shi, 1998), il a été prouvé que les performances de l'algorithme EOP ne sont pas sensibles à la taille de la population, mais plutôt au taux de convergence. Sur la base de ces résultats, la taille de la population dans le présent travail est fixée à 25 particules. De plus, nous considérons des valeurs égales pour les coefficients cognitifs et sociaux pour le calcul de la vitesse de ces particules. Dans les premières études sur l'OEP, le paramètre de poids d'inertie m avait une valeur constante. Cependant, les résultats expérimentaux indiquent qu'il est préférable de l'initialiser à une grande valeur, puis de la réduire progressivement pour obtenir des solutions raffinées. Par conséquent, m peut suivre une fonction linéairement décroissante comme définit dans l'équation (4.6) (Xin *et al.*, 2009), où m_{max} et m_{min} sont les valeurs de poids d'inertie maximale et minimale, fixées dans nos simulations à 0,9 et 0,4 respectivement. Afin de garder un temps d'exécution acceptable de l'algorithme, nous fixons le nombre maximal d'itérations $\theta_{max} = 250$. Nous avons en outre réalisé plusieurs expériences en faisant varier le nombre maximum d'itérations et le nombre de particules. Cependant, nous avons conclu que l'ajout de plus

d'itérations ou de particules n'améliore pas les performances de notre algorithme pour les scénarios étudiés.

$$m = m_{max} - \frac{m_{max} - m_{min}}{\theta_{max}} \times \theta, \tag{4.6}$$

4.4.2 Résultats des simulations

Dans cette section, nous étudions les effets des comportements des nœuds du fog sur les performances de nos algorithmes d'allocation de ressources. Ainsi, l'impact de la mobilité, de la disponibilité et de la charge des fonctions principales des nœuds du fog est étudié.

Mobilité des nœuds du fog

Dans la première série de simulations, nous étudions l'effet de la mobilité des nœuds du fog sur l'allocation des ressources pour les conteneurs déployant les applications **A1** et **A2**. Pour ce faire, nous adoptons trois scénarios avec différents modèles de mobilité : (1) modèle statique (5 RSU), (2) modèle à mobilité lente (80 téléphones portables) et (3) modèle à mobilité rapide (40 véhicules). De plus, nous considérons de traces de mobilité réélles de téléphones portables intelligents et de voitures. Les schémas de mobilité des téléphones portables sont extraits des traces de piétons, collectées auprès de volontaires à Manhattan. Leurs déplacements en ville peuvent inclure leurs circulation en métro, en bus et la plupart du temps à pied (Injong *et al.*, 2009). De plus, nous considérons des traces de mobilité de voitures extraites des déplacements de plus de 100 taxis à Rome (Bracciale *et al.*, 2014).

Dans le but de se concentrer sur l'effet de la mobilité des nœuds du fog sur les performances de l'allocation des ressources au conteneurs, le nombre de nœuds

dans chaque scénario est fixé de sorte que la quantité totale de vCPU et de RAM disponible dans les trois scénarios soit la même.

De plus, nous supposons dans ces simulations que les fonctions principales des nœuds du fog ont une charge négligeable. D'autres simulations, détaillées dans la sous-section 4.4.2, sont dédiées à la variation du nombre de nœuds pour différents scénarios.

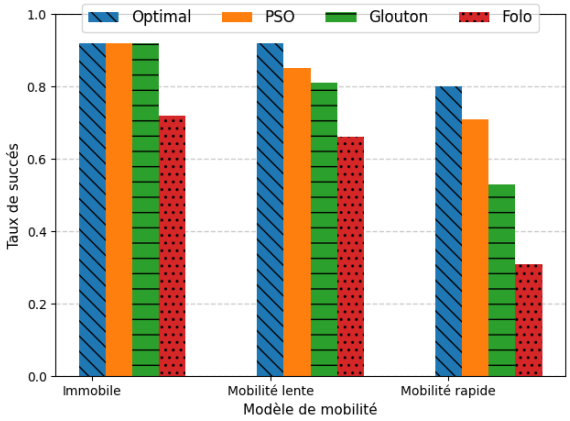
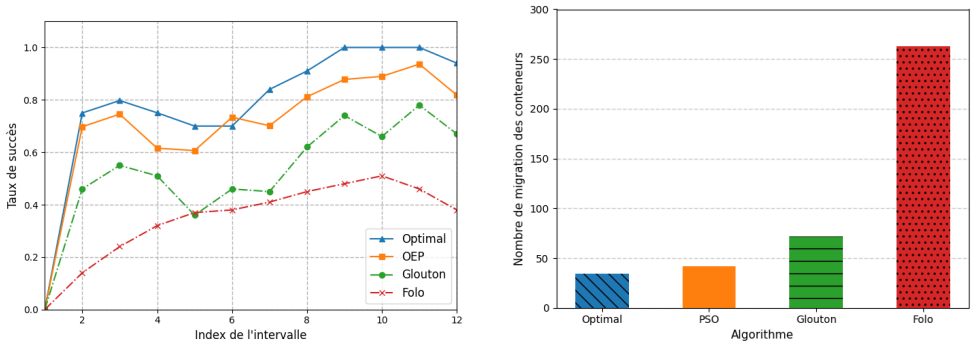


FIGURE 4.2 Ratio du succès vs. Mobilité

La figure 4.2 présente le ratio des requêtes des utilisateurs satisfaits, défini comme le nombre de requêtes servies dans un délai de réponse inférieur au seuil exigé par l'application divisé par le nombre total de requêtes au fil du temps, pour les trois scénarios de mobilité. Pour le scénario statique, les algorithmes glouton et celui basé sur OEP atteignent des performances optimales. Le ratio de réussite optimal est de 92%, où toutes les demandes des utilisateurs sont servies dans un délai inférieur au seuil déterminé pour chaque application, sauf dans le premier créneau horaire où les conteneurs sont lentement déployés dans les nœuds du fog. Pendant ce temps, l'algorithme de Folo atteint un taux de réussite inférieur de 20% par rapport à la solution optimale. La raison en est que Folo exécute à chaque itération une stratégie basée sur OEP pour placer les nouvelles tâches et celles migrées dans

les nœuds du fog. En cas de violation de contraintes, leur stratégie ne prend pas en compte la réparation des particules, ce qui impacte les performances de la convergence.

Dans le scénario de mobilité lente, le taux de réussite optimal est le même que celui du scénario statique. En effet, étant donné que les distances entre les nœuds du fog et les caméras ne varient pas de manière significative, la stratégie optimale de déploiement des conteneurs place les conteneurs sans exiger des migrations supplémentaires. Cependant, les performances des autres algorithmes se dégradent de 5%-10%. Par rapport au premier scénario, l'algorithme glouton et celui basé sur l'OEP ne sont pas en mesure d'obtenir des placements optimaux, et subissent donc plus de migration de conteneurs.



(a) Ratio des requêtes satisfaisantes vs. temps

(b) Nombre total de migrations

FIGURE 4.3 Performances du scénario à mobilité rapide

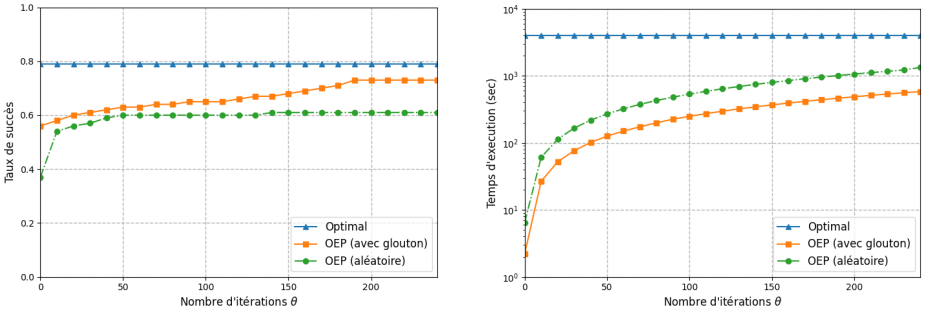
La Figure 4.3 illustre les performances des algorithmes proposés dans le scénario à mobilité rapide. Dans ce scénario, le taux de réussite de la solution optimale se dégrade de près de 15% par rapport aux scénarios décrits précédemment et nécessite un plus grand nombre de migrations de conteneurs entre les nœuds. En outre, l'algorithme glouton et celui basé sur l'OEP présentent des performances

inférieures de 10% et 30% par rapport aux performances optimales. Finalement, l'algorithme Folo atteint 72,5% de performances en moins que es performances optimales.

Dans la Figure 4.3(a), le taux de réussite des requêtes au fil du temps est présenté. L'algorithme glouton et celui basé sur l'OEP suivent la même tendance que la solution optimale en termes de taux de réussite, dicté par les distances entre les nœuds de fog et les caméras. En effet, les baisses de performances dans certains intervalles peuvent s'expliquer par la dynamique de la mobilité des nœuds du fog, qui ont tendance à s'éloigner des caméras. Enfin, les performances de Folo s'améliorent avec le temps, mais elles ne dépassent jamais 50%. Cela peut s'expliquer par le fait que Folo s'appuie sur les distances moyennes entre les nœuds du fog et les appareils finaux (caméras) pour placer les nouveaux conteneurs et ceux qui sont migrés à chaque intervalle. De plus, Folo n'exécute pas de stratégie de réparation lorsque les contraintes sont violées, ce qui dégrade d'avantage sa convergence.

La Figure 4.3(b) illustre le nombre total de migrations des conteneurs. La solution optimale présente le plus petit nombre de migrations. En effet, cette solution prévoit à l'avance le déploiement des conteneurs, en tenant compte des futurs emplacements des nœuds du fog. En outre, même si l'algorithme basé sur OEP effectue plusieurs migrations de conteneurs, il est capable d'atteindre des performances quasi optimales. Cela peut s'expliquer par le caractère aléatoire des positions des particules et l'approche de réparation lorsqu'une solution irréalisable est rencontrée par une particule. L'algorithme glouton exécute un petit nombre de migrations, car il privilégie l'utilisation de conteneurs existants au sein de la plateforme de fog computing, avant d'envisager la création ou la migration de conteneurs. Enfin, l'algorithme Folo exécute le plus grand nombre de migrations. Comme mentionné ci-dessus, le manque de réparation des particules et la prise en

compte des distances moyennes aux nœuds du fog affectent considérablement les performances de cet algorithme.



(a) Taux de succès vs. θ (b) Temps d'exécution vs. θ

FIGURE 4.4 Convergence de l'algorithme basé sur l'OEP

Les figures 4.4(a)-4.4(b) illustrent le comportement de convergence de l'algorithme basé sur l'OEP, pour le scénario à mobilité rapide. En effet, nous comparons la convergence de deux algorithmes basés sur l'OEP : (1) le premier initialise les particules en fonction de la solution de l'algorithme glouton, appelée "OEP (Glouton)" et (2) le second initialise aléatoirement les particules, nommée "OEP (Aléatoire)". Dans la Figure 4.4(a), à mesure que le nombre d'itérations θ augmente, OEP (Glouton) fournit de meilleurs résultats, qui sont quasi optimaux pour θ se rapprochant de 200 [1]. De plus, plus de simulations ont été menées avec un nombre maximum d'itérations θ_{max} plus élevé, mais n'ont pas entraîné d'amélioration des performances du taux de succès.

Selon la Figure 4.4(b), la résolution optimale du problème de provisionnement des tâches et de placement des conteneurs pour 12 intervalles de temps a une complexité de calcul élevée (environ 4000 secondes). En raison de sa NP-Difficulté, nous avons limité le nombre d'intervalle à 12 pour toutes les simulations étudiées

1. Dans cette section , "OEP (Glouton)" est utilisé, appelé "OEP" par rapport à d'autres algorithmes.

dans cette section. Comme le montre la Figure 4.4(b), les algorithmes basés sur l'OEP sont environ 10 à 20 fois plus rapides que l'approche optimale pour obtenir des résultats quasi optimaux. Nos résultats expérimentaux indiquent que l'initialisation des particules par la solution basée sur la solution de l'algorithme glouton, puis une déviation aléatoire de celle-ci, accélère le temps de convergence vers la solution quasi-optimale. Il convient de noter que l'algorithme glouton est exécuté en environ 90 ms, environ 100 fois moins que les algorithmes basés sur l'OEP, ce qui en fait un candidat potentiel pour la mise en œuvre dans de scénarios réalistes.

La disponibilité des nœuds du fog

La disponibilité des nœuds du fog peut varier dans le temps en raison des limitations liées à l'alimentation en électricité des nœuds et à la durée de vie de la batterie, etc. Pour illustrer l'effet de cette disponibilité variable, nous évaluons à travers la Figure 4.5 l'impact du nombre de nœuds du fog (de différents types) sur le taux de succès. Le nombre d'appareils pour chaque type est résumé dans le tableau **??**. Comme on peut le voir, le taux de succès s'améliore lorsque davantage de nœuds sont disponibles dans le système. En effet, avec de nombreux nœuds dédiés au à la plateforme du fog computing, les algorithmes sont capables de sélectionner de meilleurs emplacements pour y déployer les conteneurs permettant de mieux satisfaire les requêtes des utilisateurs.

Variation de la charge des fonctions principales

Nous étudions dans cette partie l'impact de la charge dynamique des fonctions principales des nœuds du fog. Nous supposons que la charge suit une distribution exponentielle avec le paramètre d'échelle β. Dans un réseau composé de 34 nœuds immobiles dédiés au fog (voir tableau 4.3), nous illustrons dans les Figures

TABLE 4.3 Les plateformes de Fog computing considérées

Appareil	**Cas 1**	**Cas 2**	**Cas 3**
RSUs	2	3	4
Voitures	12	18	24
Téléphones intelligents	20	30	40
Total	34	51	68

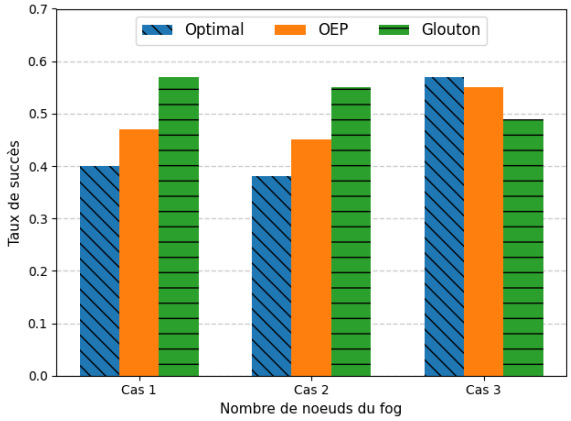
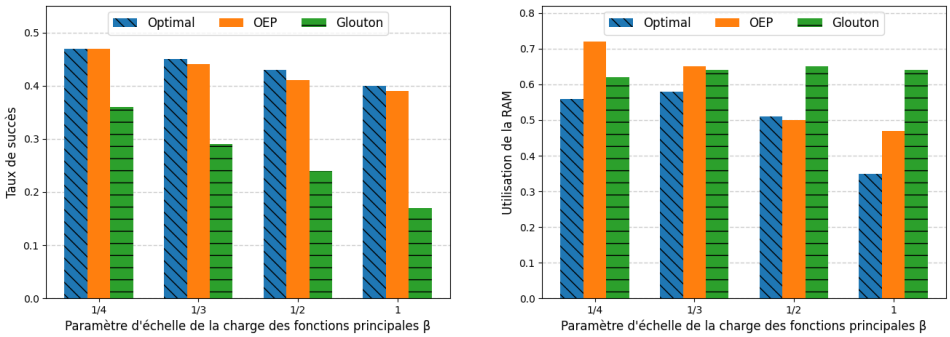


FIGURE 4.5 Taux de succès pour les différentes plateformes de Fog Computing

4.6(a)-4.6(b) l'influence du paramètre β sur le taux de réussite et les performances d'utilisation de la RAM, respectivement.

Selon la Figure 4.6(a), à mesure que le paramètre d'échelle de la charge des fonctions principales augmente, le taux de réussite diminue. En effet, avec un β plus grand, la charge des fonctions principales occupe plus de ressources. Par conséquent, les algorithmes ont moins de possibilités de placer les conteneurs dans les nœuds les plus adéquats pour satisfaire les demandes des utilisateurs dans les délais exigés par les applications. Il est à noter que la dégradation est plus importante pour l'algorithme glouton.



(a) Taux de succès vs. paramètre d'échelle β

(b) Utilisation moyenne de la RAM vs. paramètre d'échelle β

FIGURE 4.6 Effet de la variation de la charge des fonctions principales

Dans la Figure 4.6(b), l'utilisation des ressources associées au paramètre d'échelle est indiquée. Les résultats se limitent à l'utilisation de la RAM, vue que c'est la ressource qui connaît le taux d'utilisation le plus élevé. Lorsque β augmente, la solution optimale présente la plus faible utilisation de ressources dans la plupart des cas, tout en conservant le taux de réussite le plus élevé. L'algorithme glouton et celui basé sur l'OEP présentent une utilisation significative des ressources, qui diminue avec β pour l'algorithme basé sur l'OEP, alors qu'elle est stable pour l'algorithme glouton. L'algorithme basé sur l'OEP adopte la stratégie de concentration des déploiements de conteneurs dans un petit nombre de nœuds, tandis que l'algorithme glouton maintient le ratio d'utilisation de la RAM autour de 0.6, au détriment d'un plus grand nombre de nœuds convoités. Par rapport aux résultats de mobilité précédents, la variation de charge conduit les algorithmes à adopter des stratégies différentes afin d'atteindre leurs meilleures performances.

4.5 Conclusion

Dans ce chapitre, nous avons étudié conjointement les problèmes de placement des conteneurs et le problème de provisionnement des tâches pour un environnement

dynamique de *fog computing*. La mobilité des nœuds et la variation sporadique de la charge des fonctions principales sont prises en compte. Nous avons proposé une méta-heuristique basée sur l'Optimisation par essaims particulaires et une heuristique gloutonne pour résoudre le problème d'optimisation étudié. Ces solutions sont évaluées par des simulations sur un système routier intelligent déployé dans un environnement de *fog computing* dynamique et en utilisant des données réelles de mobilité. Nous avons montré que l'algorithme basé sur le OEP obtient des résultats quasi-optimaux, mais avec des temps d'exécution plus longs que l'algorithme glouton. En effet, l'algorithme glouton obtient des résultats inférieurs de 10% à 30% par rapport aux résultats optimaux avec un temps d'exécution négligeable. Enfin, nous avons étudié l'impact de la variation de la charge des fonctions de base. Cependant, la variation des ressources dans les nœuds du *fog*, due à la charge variable des fonctions de base, peut dégrader significativement les performances du système.

CHAPITRE V

MÉTHODE *ONLINE* CENTRALISÉE

5.1 Introduction

Le problème formalisé dans le chapitre 3 est celui de l'allocation de ressources et de provisionnement de tâches conjointement dans une plateforme de FC dynamique. Dans ce chapitre, nous nous intéressons seulement au problème d'allocation des ressources pour les conteneurs exécutant les applications des utilisateurs dans une plateforme de *fog computing* dynamique.

Nous introduisons le chapitre avec une mise à jour du problème formulé à la section 3.4. En outre, nous assumons qu'un conteneur ne peut être dédié qu'à un seul utilisateur simultanément. Et, nous nous focalisons sur le placement de ces conteneurs dans les nœuds du *fog*. Puis, nous présenterons notre solution *online* centralisée, basée sur l'apprentissage machine. Finalement, nous évaluerons les performances de notre solution.

5.2 Modélisation et formulation

Dans la plateforme de *fog computing* considérée, les utilisateurs finaux envoient des requêtes pour l'exécution de tâches au contrôleur de domaine qui leur est associé. Ce dernier, ayant connaissance de l'état actuel et futur des nœuds dédiés au fog du domaine, décide de la stratégie d'allocation des ressources à adopter

visant à maximiser le nombre de requêtes satisfaisantes. Ensuite, ce contrôleur envoie des demandes de déploiement de conteneurs aux nœuds du fog désignés afin d'héberger les applications des utilisateurs finaux. Ces conteneurs exécutent les tâches demandées par les utilisateurs finaux et répondent à leurs requêtes. Pour simplifier notre modèle, nous supposons qu'un conteneur répond aux requêtes d'un seul utilisateur final. Finalement, le contrôleur de domaine surveille le délai de réponse. Ce délai est défini par la somme des délais de communication entre un nœud du *fog* et l'utilisateur desservi et le délai d'exécution des tâches. Lorsque le délai de réponse dépasse le seuil de délai indiqué par Q (et défini par une exigence de qualité de service), une migration peut être déclenchée pour maintenir un nombre maximal de demandes d'utilisateurs satisfaits.

Dans notre modèle de système, nous assumons que le temps est discrétisé en intervalles. Notre système est composé d'un domaine de fog, hébergeant les nœuds du fog $\mathcal{F}$. Les emplacements de ces nœuds sur des intervalles de temps T sont désignés par la matrice $\mathbf{F}_L$. Chaque nœud f a des ressources de types $\mathcal{R}$, limitées par des capacités maximales, telles que définies dans la matrice notée $\mathbf{F}_T$. D'autre part, nous assumons que les ressources des nœuds sont principalement dédiées à leurs fonctions principales locales, et le reste de ces ressources à la plateforme de *fog*. En raison de la variabilité de la charge des fonctions principales, les ressources disponibles pour la plateforme du *fog* fluctuent. Nous définissons par $\mathbf{F}_B$ la matrice représentant la quantité de ressources allouées aux fonctions principales locales des nœuds sur les intervalles de temps de T .

Ces nœuds seront utilisés pour déployer les conteneurs qui exécutent les demandes des applications d'un ensemble d'utilisateurs $\mathcal{U}$. Nous dénotons l'ensemble de conteneurs $\mathcal{C}$, où chaque conteneur $c \in \mathcal{C}$ nécessite une certaine quantité de ressource $r \in \mathcal{R}$, définies dans la matrice $\mathbf{C}_R$. Le contrôleur de domaine du *fog* peut déclencher le déploiement ou la destruction d'un conteneur c à l'intervalle

t. En outre, les matrices $\mathbf{C}_A$ et $\mathbf{C}_D$ représentent respectivement les requêtes de déploiement et de destruction de conteneurs. Aussi, la matrice $\mathbf{C}_L$ désigne les emplacements des utilisateurs demandant des conteneurs. Enfin, nous définissons par $\mathbf{C}_F$ la matrice binaire indiquant si un conteneur c est déployé dans le nœud f à l'intervalle de temps t ou non, tandis que la matrice $\mathbf{M}$ indique un nouveau déploiement de conteneurs ou une migration dans les nœuds au fil du temps.

5.2.1 Formulation du problème

En utilisant ce modèle de système, notre objectif consiste à déterminer la stratégie optimale de placement et de migration des conteneurs dans le but de maximiser le nombre de demandes d'utilisateurs satisfaits, en ce qui concerne les exigences de délais de réponse et de limites de ressources.

Le problème associé s'écrit comme suit :

$$\max_{\mathbf{C}_F} \quad \sum_{t=1}^{T}\sum_{c=1}^{|\mathcal{C}|}\sum_{f=1}^{|\mathcal{F}|}\mathbf{S}(c,f,t) \tag{P1}$$

$$\text{s.t.} \quad Q - d(c,f,t) \leq G(\mathbf{S}(c,f,t)-1),\ c\in\mathcal{C}, f\in\mathcal{F}, 1\leq t\leq T, \tag{P1.a}$$

$$\mathbf{S}(c,f,t) \leq \mathbf{C}_F(c,f,t),\ c\in\mathcal{C}, f\in\mathcal{F}, 1\leq t\leq T, \tag{P1.b}$$

$$\mathbf{C}_F(c,f,t) \leq \sum_{k=1}^{t}\mathbf{C}_A(c,k) - \sum_{k=1}^{t}\mathbf{C}_D(c,k),\ c\in\mathcal{C}, 1\leq t\leq T, \tag{P1.c}$$

$$\sum_{f\in\mathcal{F}}\mathbf{C}_F(c,f,t) \leq 1,\ c\in\mathcal{C}, 1\leq t\leq T, \tag{P1.d}$$

$$\mathbf{M}(c,f,t) \geq \mathbf{C}_F(c,f,t) - \mathbf{C}_F(c,f,t-1),\ c\in\mathcal{C}, f\in\mathcal{F}, \tag{P1.e}$$

$$\mathbf{M}(c,f,1) \geq \mathbf{C}_F(c,f,1), \forall c\in\mathcal{C}, f\in\mathcal{F}, \tag{P1.f}$$

$$\gamma(f,r,t) \leq 1,\ f\in\mathcal{F}, r\in\mathcal{R}, 1\leq t\leq T, \tag{P1.g}$$

où $\mathbf{S}$ est une matrice binaire indiquant si l'exigence en terme de délai est

respectée (i.e. délai de réponse $\leq$ Q), et G est un grand nombre positif sélectionné aléatoire. Aussi, $d(c, f, t)$ est le délai de service du conteneur c déployé dans FN f dans le créneau horaire t, exprimé par l'Équation (5.2), où d_1, d_2 et d_3 sont respectivement le délai de communication, le délai de déploiement du conteneur et le délai de calcul. d_1 est donné par $d_1(c, f, t) = 2d_{\text{tx}}(c, f, t)\mathbf{C}_F(c, f, t)$, où $d_{\text{tx}}(c, f, t) = \frac{L_c}{B \log_2(1+\text{P}||\mathbf{C}_\text{L}(\text{c,t})-\mathbf{F}_\text{L}(\text{f,t})||^{-\alpha}/\sigma)}$ est le délai de liaison sans fil entre l'utilisateur demandant le conteneur c dans le temps t et nœud f, L_c est la taille d'un paquet pour le conteneur c, B est la bande passante de transmission, P est la puissance de transmission, α est l'exposant d'atténuation σ est la puissance de bruit et $||.||$ est la norme euclidienne.

$$d(c, f, t) = d_1(c, f, t) + d_2(c, f, t) + d_3(f, t) \tag{5.2}$$

Le délai de déploiement d'un conteneur d_2 peut être exprimé par $d_2(c, f, t) = d_{\text{con}}(c)\mathbf{M}(c, f, t)$, où $d\ _{rmcon}(c)$ est le temps nécessaire pour déployer le conteneur c (temps requis pour le téléchargement et l'installation des paquets de l'application demandée). Enfin, le délai de calcul d_3 dépend des capacités du nœud.

Dans (P1.g), $\gamma(f, r, t)$ est le taux d'utilisation de la ressource r par le nœud f à l'intervalle t, exprimé dans l'Equation 5.3.

$$\gamma(f, r, t) = \frac{\sum_{c=1}^{|\mathcal{C}|} \mathbf{C}_F(c, f, t) \cdot \mathbf{C}_R(c, r)}{\mathbf{F}_T(f, r) - \mathbf{F}_B(f, r, t)}. \tag{5.3}$$

Les contraintes (P1.a)-(P1.b) garantissent que seuls les délais de réponse des conteneurs déployés et qui respectent l'exigence QoS pour le délai Q sont considérés dans la mise à jour de la matrice S. Les contraintes (P1.c)-(P1.d) garantissent qu'un conteneur est placé sur un seul nœud entre l'intervalle de la

demande de déploiement et l'intervalle de la demande de destruction. En outre, les contraintes (P1.e)-(P1.f) s'assurent qu'un nouveau déploiement/migration est proportionnel à la différence entre les placements de conteneurs aux temps t et $t-1$. Finalement, la contrainte (P1.g) garantit que les capacités en termes de ressources des nœuds du *fog* sont respectées lors du déploiement de conteneurs.

Le problème (P1) peut être prouvé comme étant NP-Difficile. En effet, le cas particulier du problème, où un seul intervalle de temps t et une seule ressource r sont considérés est analogue au problème d'affectation généralisé (Generalized Assignment Problem) bien connu (Ross et Soland, 1975). Puisque GAP est déjà prouvé NP-difficile, alors par restriction le cas particulier de (P1) est également NP-difficile (Garey et Johnson, 2002).

5.3 Algorithme d'allocation de ressources basé sur l'apprentissage profond par renforcement

En raison de la NP-Difficulté du Problème (P1) et de la nature dynamique de l'environnement de *fog computing*, le problème d'allocation des ressources devient très difficile à résoudre. Quand ce problème est étudié à large échelle, pour des cas d'utilisation réalistes, il devrait être insoluble d'une manière optimale. Par conséquent, nous introduisons une plateforme basée sur l'apprentissage profond par renforcement (DRL pour Deep Reinforcement Learning en anglais) (Watkins et Dayan, 1992; Mnih *et al.*, 2015; Mnih *et al.*, 2013) pour la résolution de ce problème. La motivation vient de la sous-optimalité obtenue par les solutions heuristiques, en particulier lors de leur exécution en ligne, avec une connaissance partielle ou nulle des conditions des environnements étudiés.

5.3.1 Concepts de l'apprentissage par renforcement profonds

Avant de présenter notre approche basée sur le DRL, nous fournissons une introduction générale à l'apprentissage profond par renforcement.

En apprentissage automatique, l'apprentissage par renforcement (RL, pour "Reinforcement Lerning" en anglais) est un processus dans lequel un agent optimise son comportement en interagissant avec son environnement (Sutton et Barto, 1998). Contrairement aux approches classiques de contrôle optimal, le RL ne demande pas systématiquement la connaissance d'un modèle précis des dynamiques de l'environnement. À chaque étape, l'agent RL exécute une *action* qui produit une récompense, qui lui donne une indication sur la qualité de cette action. La fonction qui indique l'action à entreprendre dans un certain état s'appelle politique. L'objectif principal de l'agent est de trouver une politique qui maximise la récompense totale cumulée à travers les étapes. Cela implique que les récompenses définissent implicitement le comportement de l'agent. Dans le RL, les agents apprennent à agir par "essais et erreurs", en améliorant progressivement leurs performances au fur et à mesure que l'apprentissage progresse.

Classiquement, l'apprentissage par renforcement repose sur un processus de décision markovien (MDP, pour "Markov Decision Process" en anglais), qui propose un cadre pour le problème d'apprendre à réaliser un but. Il est défini comme $\mathcal{M} = < S , A , P , \pi , R , \gamma >$ et est composé d'un espace d'état fini S, un espace d'action A, un ensemble de récompense R, un ensemble de probabilité de transition P reliant chaque paire état-action $(s, a) \in S \times A$ vers l'état suivant $s' \in S$, une politique $\pi : S \to A$, une fonction de récompense $r : S \times A \to R$, et un facteur d'actualisation $\gamma \in (0, 1]$ qui pénalise les récompenses futures et privilégier les récompenses immédiates.

L'interaction entre l'agent et l'environnement est un processus continu. En effet,

à chaque époque de décision κ, l'agent collecte l'état du système s_κ et sélectionne une action a_κ selon une politique $\pi(s_\kappa)$. Après avoir effectué a_κ, le système passe à un nouvel état $s_{\kappa+1}$ et l'agent calcule la récompense résultante $r_\kappa = r(s_\kappa, a_\kappa)$. L'objectif de l'agent est de maximiser la récompense cumulée dans le temps $\bar{r}_0 = \mathbb{E}[\sum_{\kappa=0}^{K} \gamma^\kappa r_\kappa]$, où K est le nombre maximal d'époques.

L'algorithme *Q-learning*

Le *Q-learning* (Watkins et Dayan, 1992) est l'une des techniques d'apprentissage par renforcement les plus pratiques, car il n'a pas besoin de la formulation complète de la transition. Cela rend la méthode plus réaliste car l'agent est confronté à l'état actuel de l'environnement de *fog* et non à celui qui a été précédemment conçu. De plus, l'algorithme *Q-learning* présente un avantage en termes de rapidité de calcul, ce qui est adéquat pour une prise de décision rapide dans les systèmes de *fog computing* (Watkins et Dayan, 1992). Dans un tel algorithme, la valeur de l'action a dans l'état s est indiquée par la valeur d'action $Q(s_\kappa; a_\kappa)$. Le *Q-learning* construit progressivement des informations sur les meilleures actions à entreprendre dans chaque état possible. La fonction de la valeur d'action optimale $Q^*(s, a)$ représente le rendement réalisable maximal attendu en suivant la meilleure politique, et est définie comme suit

$$Q^*(s, a) = max_\pi \mathbb{E}[R_\kappa | s_\kappa = s, a_\kappa = a, \pi] \tag{5.4}$$

Ainsi, à une époque donnée κ, la politique optimale correspond à la sélection de l'action qui maximise $Q(s_\kappa, a_\kappa)$. Trouver la politique optimale revient à obtenir une estimation de $Q(s_\kappa, a_\kappa)$ qui se rapproche le plus de sa valeur réelle. L'estimation de $Q(s_\kappa, a_\kappa)$ peut être mise à jour à chaque époque comme suit

$$Q(s_\kappa, a_\kappa) = Q(s_\kappa, a_\kappa) + \alpha[r_{\kappa+1} + \gamma \max_a Q(s_{\kappa+1}, a_\kappa) - Q(s_\kappa, a_\kappa)] \tag{5.5}$$

où α est le taux d'apprentissage et γ est le facteur d'actualisation.

En général, il existe de nombreuses manières possibles permettant de sélectionner des actions dans le RL. Ces techniques alternent deux étapes représentant l'exploitation et l'exploration. L'exploitation dans la procédure d'apprentissage représente l'étape où l'agent prend des mesures optimales en fonction de ses connaissances. Tandis que l'étape d'exploration permet à l'agent de négliger les résultats optimaux obtenus jusque là en vue de raffiner la politique existante en sélectionnant des actions aléatoires. Dans cette étude, nous utilisons l'algorithme $\epsilon-greedy$ (Tokic et Palm, 2011) pour savoir quand appliquer l'action d'exploration ou d'exploitation dans le processus d'apprentissage. Avec une petite probabilité ϵ, l'agent sélectionne une action au hasard, et suit la meilleure politique qu'il a construit jusqu'à cette étape pour le reste du temps.

Apprentissage par renforcement profond

De toute évidence, la grande taille de l'espace d'états S et de l'espace d'action A vont entraîner une grande taille de la matrice Q. Pour résoudre ce problème, l'abstraction de cette matrice et son stockage dans un réseau de neurones est considéré comme une solution efficace (Mnih *et al.*, 2013). Le réseau profond Q (DQN pour "Deep Q-Network" en anglais) est un réseau neuronal multi-couche utilisé pour estimer les valeurs de la fonction Q. L'entrée pour le réseau est l'état actuel de l'environnement s, tandis que la sortie est la valeur Q correspondante pour chacune des actions $a \in A$. Par conséquent, le DQN dérive la corrélation entre chaque paire état-action (s, a) du système sous contrôle et sa fonction $Q(s, a)$.

Par conséquent, la technique d'apprentissage profond *Q-learning* considérée est

composée d'une phase de construction du DQN et d'une phase d'apprentissage de Q. Dans le DQN, la matrice Q est remplacée par un réseau de neurones Q avec des poids θ, utilisé pour stocker efficacement les informations de la matrice Q.

Afin de construire un DNN suffisamment précis, nous devons accumuler suffisamment d'échantillons d'estimations des valeurs $Q(s, a)$ et de paires (s, a) correspondants. Pour l'allocation des ressources du fog, nous générons aléatoirement des demandes de déploiement de conteneurs pour obtenir suffisamment d'états et d'estimations de la valeur $Q(s, a)$.

L'objectif de l'entraînement consiste à minimiser la fonction de perte $L(\theta)$, qui est définie comme suit :

$$L(\theta) = \mathbb{E}[(y_\kappa - Q(s_\kappa, a_\kappa; \theta))^2] \tag{5.6}$$

Où y_κ est un autre réseau cloné du réseau principal pour calculer les valeurs cibles (Mnih *et al.*, 2015) défini dans l'Équation (5.7). L'avantage d'utiliser ce deuxième réseau est qu'à chaque étape de l'entraînement, les valeurs de Q changent, et si nous utilisons un ensemble de valeurs en constante évolution pour ajuster les valeurs de ce réseau, alors les estimations des valeurs peuvent devenir ingérables. Afin d'atténuer ce risque, les poids des réseaux cibles θ' sont fixes et mis à jour mis à jour toutes les τ étapes à partir du réseau d'origine.

$$y_\kappa = r(s_\kappa, a_\kappa) + \gamma Q\left(s_{\kappa+1}, \pi(s_{\kappa+1}|\theta); \theta^-\right), \tag{5.7}$$

D'autre part, pour lutter contre l'instabilité du DNN, DQN introduit aussi la technique de relecture des expériences. Cette technique consiste à stocker les profils de transition, définis comme $e_\kappa = (s_\kappa, a_\kappa, r_\kappa, s_{\kappa+1})$ dans une mémoire de capacité N_D. L'utilisation de ces transitions permet d'améliorer le processus

d'apprentissage. En effet, en apprenant directement à partir de transitions consécutives créé une corrélations entre ces échantillon. De ce fait, utiliser les transitions stockés dans la mémoire de relecture permet de rompre ces corrélations et lisse ainsi l'apprentissage.

5.3.2 Algorithme basé sur l'apprentissage par renforcement profond

Nous présentons ici l'approche d'allocation de ressources basée sur le DRL proposée pour résoudre le problème (P1). Un agent DRL centralisé collecte périodiquement l'état de la plateforme de *fog computing*. Ensuite, à chaque demande de déploiement de conteneur, l'agent détermine le nœud où celui-ci devrait être déployé en utilisant l'approche proposée puis l'exécute.

Dans notre approche, à chaque époque κ, une nouvelle demande de déploiement de conteneur est considérée. En fonction de l'état des nœuds présents et de leur futur comportement, l'agent entreprend une action pour placer le conteneur dans un nœud spécifique, afin de maximiser le nombre total de conteneurs dont le délai de service ne dépasse pas le seuil prédéfinit.

Les états, les actions et les récompenses de l'agent de DRL sont définit comme suit :

L'état

Dans notre modèle, l'espace des états est défini par les requêtes de déploiement de conteneur et le statut des nœuds. Un état s dans une époque κ est décrit comme suit : $s_\kappa = \{\mathbf{c}_R^\kappa, \mathbf{c}_L^\kappa, G_\kappa, \mathbf{F}_R^\kappa, \mathbf{F}_L^\kappa\}$, où

- $\mathbf{c}_R^\kappa$ de taille $1 \times |\mathcal{R}|$ définit les besoins en ressources du conteneur à déployer à l'époque κ,
- $\mathbf{c}_L^\kappa$ dénote la localisation dans un espace cartésien de l'utilisateur

demandant le déploiement du conteneur à l'époque κ,

— G_κ est la durée de vie du conteneur à déployer à l'époque κ en termes d'intervalles,

— $\mathbf{F}_R^\kappa$ de taille $|\mathcal{F}| \times |\mathcal{R}|$ définit les ressources dans les nœuds disponibles à l'époque κ, et peut être calculé en utilisant $\mathbf{F}_T$ et $\mathbf{F}_B$,

— $\mathbf{F}_L^\kappa$ de taille $2 \times |\mathcal{F}|$ présente les localisations dans un espace cartésien des nœuds à l'époque κ.

Il est important de noter que le nombre total d'époques K correspond au nombre total de demandes de déploiement de conteneurs durant tous les intervalles T, c'est-à-dire $K = \sum_{t=1}^{T} \sum_{c=1}^{|\mathcal{C}|} \mathbf{C}_A(c,t)$, et que le nombre d'époques à l'intervalle t est égal au nombre de demandes de déploiement de conteneur dans t, défini comme $n(t) = \sum_{c=1}^{|\mathcal{C}|} \mathbf{C}_A(c,t)$. Par conséquent, à chaque époque, une seule demande de conteneur est reçue et servie.

L'action

À l'époque κ, l'action de l'agent de DRL pour l'allocation des ressources de l'environnement de *fog* a_κ correspond à l'indice du nœud (c'est-à-dire $a_\kappa \in \mathcal{F}$) où le conteneur demandé doit être déployé. Le conteneur peut également être déployé dans la couche cloud (notée par $a_\kappa = c_l$).

L'espace des actions est défini comme $A = \mathcal{F} \cup \{c_l\}$.

La récompense

La récompense à l'époche κ est définit comme suit :

$$r_\kappa = \sum_{j=1}^{G_\kappa} \mathbf{S}(c_\kappa, f_\kappa, j), \tag{5.8}$$

où c_κ et f_κ représentent le conteneur et le nœud sélectionné pour le déployer à l'époque κ respectivement. r_κ reflète le conteneur c_κ répond aux requêtes de

ses utilisateurs dans un délai inférieur à Q pendant les intervalles G_κ, tout en respectant la contrainte (P1.a).

La solution basée sur le DRL proposée dans cette section est formellement présentée dans l'Algorithme 3.

Algorithme 3 Algorithme basé sur le DRL

1: Initialiser la mémoire de relecture $\mathcal{D}$
2: Initialiser la fonction de action-valeur Q avec des poids aléatoires
3: **pour** episode $= 1 \dots M$ **faire**
4: Initialiser la séquence s_1
5: **pour** $\kappa = 1 \dots K$ **faire**
6: Mettre en place l'état $s_\kappa = \{\mathbf{c}_R^\kappa, \mathbf{c}_L^\kappa, G_\kappa, \mathbf{F}_R^\kappa, \mathbf{F}_L^\kappa\}$
7: Avec une probabilité ϵ selectionner une action aléatoire $a_\kappa \in A$, sinon sélectionner $a_\kappa = \max_a Q^*(s_\kappa, a; \theta)$
8: Éxecuter l'action a_κ et observer la récompense r_κ, comme défini dans (5.8), et le nouvel état $s_{\kappa+1}$
9: Conserver la transition $e_\kappa = (s_\kappa, a_\kappa, r_\kappa, s_{\kappa+1})$ dans $\mathcal{D}$
10: Échantillonner un mini-ensemble aléatoire de transitions (s_j, a_j, r_j, s_{j+1}) de $\mathcal{D}$ et entraîner le réseau
11: $y_j = \begin{cases} r_j, \text{si l'etat } s_{j+1} \text{ est final} \\ rj + \gamma \max_a Q(s_{j+1}, a; \theta^-), \text{sinon.} \end{cases}$
12: Effectuer une étape de descente de gradient avec la perte $L(\theta)$ (eq. (5.6))
13: Mettre à jour le vecteur de poids θ^- chaque τ itérations
14: **fin pour**
15: **fin pour**

5.4 Évaluation des performances

Dans cette section, nous décrivons les paramètres de la simulation, puis les performances de notre solution sont évaluées pour différents scénarios, et comparées aux solutions de l'état de l'art.

En outre, nous étudions l'effet de la variation de charge des utilisateurs et de la mobilité des nœuds sur les performances de la solution proposée dans ce chapitre.

5.4.1 Paramètres des simulations

Environnement de simulation

Dans cette section, nous allons considérer le même cas d'utilisation de système routier intelligent déployé dans un environnement de *fog computing* dynamique, énoncé dans la section 4.4. Les paramètres de l'environnement de simulation sont résumés dans le Tableau 5.1. Nous définissons la capacité des ressources des nœuds comme décrit dans le Tableau 4.1 ainsi que les applications définit dans le Tableau 4.2. Dans notre système, les utilisateurs finaux uniformément distribués envoient des demandes de déploiement de conteneur au contrôleur de domaine du *fog* pour traiter les tâches de leurs applications, suivant une distribution de Poisson ayant un taux d'arrivée moyen λ.

Table 5.1 Paramètres de l'environnement de simulation

Paramètre	**Valeur**
Zone géographique	$700m^2$
Taille du paquet	1500 bits
Coefficient d'atténuation (α)	3
Puissance bruit gaussien (σ)	$0,02$ Watts
Bande passante (B)	20 MHz

Paramètres de l'apprentissage profond

Les performances de plusieurs algorithmes d'apprentissage en profondeur dépendent des caractéristiques choisis à l'entrée et des valeurs des hyperparamètres (Neary, 2018). Cela nécessite généralement d'explorer un vaste espace d'architectures de réseaux de neurones et de valeurs des hyperparamètres (Ranjit

et al., 2019). Pour résoudre ce problème, nous avons considéré une solution déployé dans une infrastructure de *cloud computing*, constituée des outils Kubeflow (Kubeflow, 2020b) et Katib (Kubeflow, 2020a). Ces outils recherchent d'une manière automatisé la meilleur architecture de réseau de neurones à adopter et les meilleurs valeurs pour les hyperparamètres.

Sur la base des expériences menées à l'aide de la solution susmentionnée, nous avons constaté que les meilleures performances pour l'architecture DNN centralisée en termes de convergence sont obtenues lors de l'utilisation d'un réseau de neurones à deux couches entièrement connecté, avec des couches cachées composées de 128 neurones ReLu. Outre la recherche de la meilleure architecture de réseau de neurones, nous avons également mené plusieurs expériences afin de déterminer certaines valeurs d'hyperparamètres. Plus précisément, nous avons expérimenté plusieurs valeurs de capacité de mémoire tampon de relecture allant de 5000 à 10000, la taille du mini-ensemble de transitions comprise entre 128 et 512 transitions et un taux d'apprentissage allant de 10^{-6} à 10^{-4}. Le facteur actualisation a été initialisé à $\gamma = 0.9$.

5.4.2 Résultats des simulations

Nous étudions ici les effets de la variation de la charge des utilisateurs finaux et de la mobilité des nœuds du *fog* sur les performances de notre algorithme d'allocation de ressources basé sur le DQN. Ensuite, nous les comparons à deux méthodes de l'état de l'art :

— la stratégie d'allocation du nœud le plus proche (NF), où le nœud le plus proche disponible est sélectionné pour répondre à la demande de l'utilisateur,

— la stratégie d'allocation aléatoire du nœud (RF), où un nœud disponible dans la portée de communication de l'utilisateur est assigné aléatoirement

pour répondre à la demande de ce dernier.

Les performances de ces algorithmes sont comparées à celles de la solution optimale, qui est la solution du problème formulée dans la sous-section 5.2.1 et obtenue à l'aide de IBM CPLEX (CPLEX, 2016b). .

Effet de la charge des requêtes des utilisateurs

Dans ce paragraphe, nous étudions l'impact de λ sur la performance du taux de réussite. Ce taux est définit comme étant le ratio de toutes les requêtes répondues dans un délai inférieur au seuil prédéterminé (que nous appellerons aussi satisfaisantes) par rapport à toutes les requêtes reçu par le contrôleur. Pour notre algorithme, plus de 250K épisodes sont utilisés pour l'apprentissage, où chaque épisode est composé d'une configuration de 30 intervalles. La Figure 5.1 présente le comportement d'apprentissage de notre algorithme pour des valeurs de λ qui varient de 20 à 40. Comme on peut le constater, notre algorithme converge vers sa politique optimale après 100K itérations pour $\lambda = 20$. Par ailleurs, lorsque λ augmente, la convergence devient plus lente et nécessite un plus grand nombre d'épisodes. Pour $\lambda = 30$, le modèle converge vers sa politique optimale en 200K itérations. Cependant, pour $\lambda = 40$ la convergence est plus lente, et n'a pas été atteinte pour les 250K itérations. Cela peut s'expliquer par le fait qu'un λ plus élevé (c'est-à-dire un nombre de requêtes plus important) génère un système de plus grande dimension qui nécessite plus d'échantillons d'apprentissage ou d'épisodes.

La figure 5.2 illustre les performances de l'algorithme proposé basé sur le DQN, de l'algorithme NF, de l'algorithme RF et de la solution optimale, en fonction de λ. Notre algorithme atteint des performances quasi optimales, tout en surpassant les algorithmes de l'état de l'art. Ceci est principalement dû à l'intelligence du DQN par rapport aux approches simples NF et RF. De plus, de meilleures performances

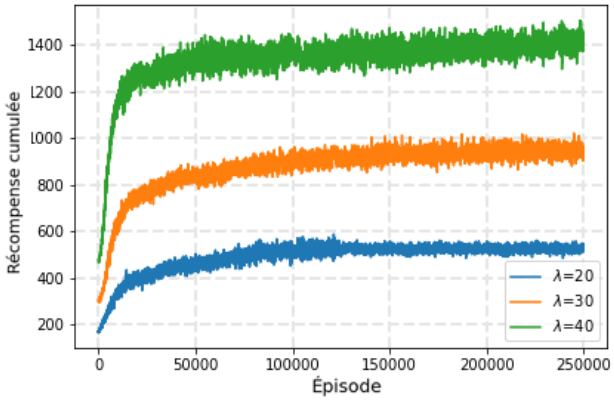


FIGURE 5.1 Récompense cumulée vs. nombre d'épisodes (différentes valeurs de λ)

pourraient être obtenues pour $\lambda \geq 40$ lorsque le modèle est entraîné avec plus de 250K épisodes.

Effet de la mobilité des nœuds

Dans ce paragraphe, nous évaluons les performances des algorithmes pour différents scénarios de mobilité, où le taux moyen d'arrivée des requêtes des utilisateurs est fixé à $\lambda = 20$. Nous adoptons quatre scénarios avec différents modèles de mobilité :

- Nœuds du *fog* immobiles (5 RSU)
- Nœuds du *fog* à mobilité lente (40 téléphones portables)
- Nœuds du *fog* à mobilité rapide (20 véhicules)
- Nœuds du *fog* à mobilité mixte (2 RSU, 6 véhicules et 12 téléphones portables)

Afin de cerner uniquement les effets de la mobilité sur les performances de notre solution, la quantité totale de ressources disponibles est la même pour tous les scénarios.

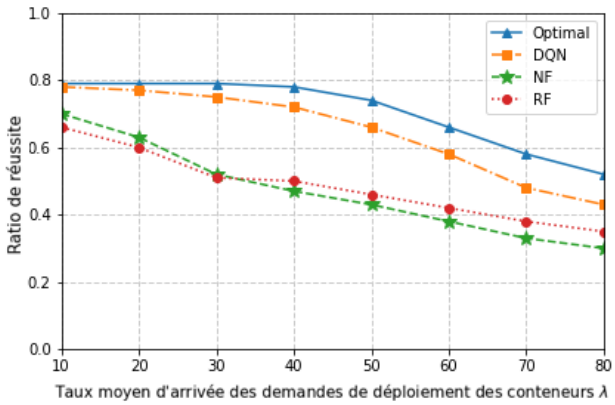


FIGURE 5.2 Taux de succès vs. taux moyen d'arrivée λ

La Figure 5.3 montre le comportement de convergence de l'algorithme proposé pour les différents scénarios de mobilité. Dans le premier scénario (Nœuds immobiles), l'algorithme proposé converge très rapidement vers la politique optimale (quelques épisodes). Ceci est attendu puisque l'espace d'action est petit et que les nœuds sont statiques. Néanmoins, le scénario de mobilité lente atteint la même récompense cumulée que le premier scénario après 50K épisodes. Cela signifie qu'un grand nombre de nœuds mobiles lents à capacité limitée peut être utilisé de manière équivalente à un petit nombre de nœuds immobiles à grande capacité. D'autre part, les scénarios de mobilité rapide et de mobilité mixte obtiennent une récompense cumulative inférieure à la valeur atteinte lors de la convergence des deux autres scénarios. Cela est principalement dû à la grande mobilité des nœuds qui limite le déploiement satisfaisant des conteneurs.

La Figure 5.4 compare les performances de l'algorithme basé sur le DQN, de l'algorithme NF, de l'algorithme RF et de la solution optimale pour différents scénarios de mobilité. Dans le premier scénario (nœuds immobiles), tous les algorithmes atteignent un taux de réussite optimal. Dans le deuxième scénario (mobilité lente), l'algorithme que nous proposons obtient des performances quasi

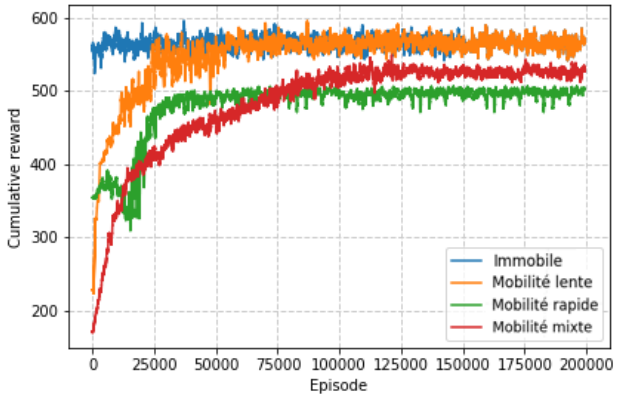


FIGURE 5.3 Récompense cumulée vs. scénario de mobilité

optimales, tandis que les algorithmes NF et RF présentent des taux de réussite dégradés de 15% et 20%, respectivement, par rapport au taux de la solution optimale. Avec une mobilité plus élevée, les performances de tous les algorithmes se dégradent davantage, avec une moindre dégradation pour l'approche proposée. Enfin, avec des schémas de mobilité mixtes, l'algorithme basé sur le DQN est proche de la solution optimale, avec une perte de seulement 10%, alors que les algorithmes NF et RF présentent une dégradation de plus de 25%.

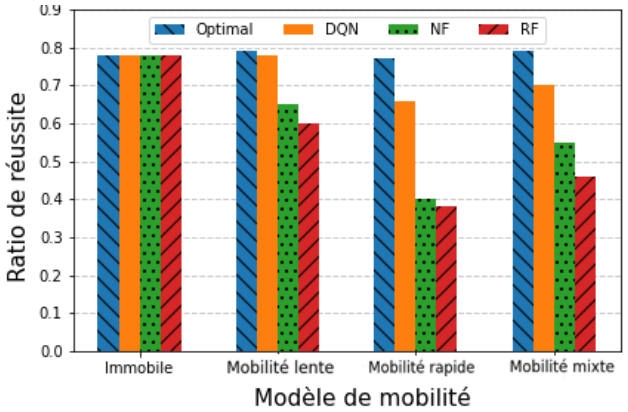


FIGURE 5.4 Taux de succès vs. scénario de mobilité

5.5 Conclusion

Dans ce chapitre, nous avons présenté notre solution *online* pour l'allocation de ressources dans un environnement de *fog computing* dynamique. Le problème de placement de conteneurs dans les nœuds du *fog* en vue de maximiser le nombre de requêtes dont le délais de réponse est inférieur à un seuil prédéterminé a été formulé sous la forme d'un programme linéaire en nombres entiers. En raison de sa NP-Difficulté, nous avons proposé un algorithme basé sur le DQN qui résout efficacement le problème. Les performances de cet algorithme sont évaluées par des simulations utilisant des données de mobilité réels. Les résultats montrent que notre solution atteint des résultats quasi-optimaux en termes de taux de satisfaction, nettement meilleurs que les approches de l'état de l'art.

CHAPITRE VI

MÉTHODE *ONLINE* COLLABORATIVE

6.1 Introduction

Comme les environnements de *fog computing* déploient généralement des applications sensibles à la latence, les algorithmes d'allocation des ressources doivent prendre des décisions immédiates et rapides pour déployer les applications dans le bon nœuds . De plus, ces algorithmes doivent être adaptés aux systèmes à grande échelle. Leur extensibilité doit être bonne.

Nous présentons ici la méthode d'allocation des ressources collaborative proposée pour résoudre le problème P1.

6.2 La méthode QMIX

6.2.1 Apprentissage par renforcement à agents multiples

Un système multi-agent est un groupe d'entités autonomes, en interaction, partageant un environnement commun, qu'ils perçoivent à l'aide de capteurs et sur lequel ils agissent à l'aide d'actionneurs (Hu *et al.*, 1998). Les systèmes multi-agents sont appliqués dans une variété de domaines, notamment la robotique, le contrôle distribué, la gestion des ressources, les systèmes collaboratifs d'aide à la décision (Busoniu *et al.*, 2008). L'apprentissage par renforcement multi-agent (MARL "Multi-Agent Reinforcement Learning" en anglais) repose

sur le même principe que le RL à un seul agent, sauf que ici les agents ont une vue restreinte de l'environnement et ne perçoivent qu'une partie locale de l'environnement globale. D'une manière similaire, chaque agent d'apprentissage perçoit l'état de son environnement local et entreprend une action, qui fait passer l'environnement global à un nouvel état. Un signal de récompense scalaire évalue la qualité de chaque transition. Ainsi, l'agent doit maximiser la récompense cumulative tout au long de cette interaction. La Figure 6.2.1 illustre le fonctionnement du MARL.

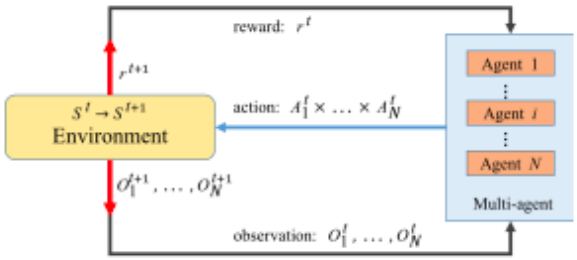


FIGURE 6.1 Apprentissage par renforcement Multi-Agent (Wen *et al.*, 2018)

6.2.2 Aperçu de la méthode QMIX

QMIX est une méthode d'apprentissage qui repose sur MARL. En tant que méthode collaborative, QMIX (Rashid *et al.*, 2018) est une approche qui se situe entre deux méthodes extrêmes, à savoir le Q-learning Indépendant (IQL pour "Independant Q-Learning" en anglais) (Tan, 1993) et le multi-agent contrefactuel (COMA pour "Conterfactual Multi-Agent" en anglais) (Foerster *et al.*, 2018). D'une part, IQL traite un problème multi-agents en le divisant en une collection de problèmes que peuvent être résolu simultanément, chaque problème est affecté à un agent. Ces agents partagent le même environnement. Mais ne partagent pas les récompenses. D'une autre part, COMA apprend une fonction état-action

entièrement centralisée Q_{tot} et puis l'utilise dans le but de guider l'optimisation de politiques décentralisées dans une plateforme actor-critic.

L'approche fondamentale de QMIX est d'apprendre une fonction action-valeur, notée $Q_{\text{tot}}(\tau, a)$, où τ représente un historique action-observation et a représente une action commune Q_{tot}. Contrairement aux réseaux de décomposition de valeurs (VDN pour "Value Decomposition Networks" en anglais) (Sunehag *et al.*, 2018) où Q_{tot} est défini comme la somme des fonctions de valeur individuelles, notée $Q_i(\tau^i, a^i)$ pour agent i, QMIX garantit qu'un *argmax* global exécuté sur Q_{tot} fournit le même résultat qu'un ensemble d'opérations *argmax* individuelles exécutées sur chaque fonction Q_i, c'est-à-dire

$$\arg\max_a Q_{\text{tot}}(\tau, a) = \begin{pmatrix} \arg\max_{a^1} Q_1(\tau^1, a^1) \\ \vdots \\ \arg\max_{a^D} Q_D(\tau^D, a^D) \end{pmatrix}. \tag{6.1}$$

Par conséquent, il suffit d'appliquer une contrainte de monotonie sur la relation entre Q_{tot} et chaque fonction Q_i comme suit :

$$\frac{\partial Q_{\text{tot}}}{\partial Q_i} \geq 0, \forall i = 1, \ldots, D. \tag{6.2}$$

L'architecture de QMIX se compose de l'ensemble de réseaux des agents représentant chaque Q_i, et d'un réseau de mixage qui les combine en Q_{tot}, non pas comme une simple somme, mais d'une manière non-linéaire complexe qui assure la cohérence entre la politique centralisée et celle décentralisée. En même temps, il applique la contrainte (6.2) en limitant le réseau de mixage à des poids positifs. Ces poids sont produits par un hyper-réseau séparé, c'est-à-dire un autre réseau de neurones, et sont conditionnés par l'état global s. QMIX est entraîné de bout en bout en minimisant la fonction de perte $L(\theta)$ définie par

$$L(\theta) = \mathbb{E}[(y_t - Q_{\text{tot}}(\tau_t, a_t; \theta))^2], \tag{6.3}$$

où θ est le vecteur de poids du réseau de neurones QMIX, et y_t est la valeur cible, défini dans l'Équation (5.7).

Par conséquent, QMIX peut être considéré comme une fonction action-valeur centralisée complexe avec une représentation factorisée qui s'adapte bien au nombre d'agents, permettant ainsi d'extraire facilement des politiques décentralisées via des opérations *argmax* individuelles en temps linéaire (Rashid *et al.*, 2018).

6.3 Algorithme collaboratif basé sur QMIX

Le problème de provisionnement de services multi-agents associé est formulé comme un jeu de Markov partiellement observable, $\mathcal{M} =< S, \{O\}^n, \{A\}^n, P, \pi, R, \gamma >$, composé d'agents D déployés dans les nœuds qui représentent les contrôleurs du *fog*, notés $d = 1, \ldots, D$, tels que $D \leq |\mathcal{F}|$. Chaque agent gère un groupe unique de N_d nœuds et peut décider si ses nœuds peuvent satisfaire la demande entrante (délai de réponse inférieur au seuil prédéfinit compte tenu de leur statut et de leurs capacités à cet instant. Si le déploiement du conteneur respecte le délai de service, l'agent décide également du nœud où le conteneur va être déployé. S, P, R, π et γ représentent respectivement l'état du système, l'ensemble des probabilités de transition, la récompense, la politique et le facteur d'actualisation, qui sont identiques à ceux définis dans la section IV. En revanche, tous les agents partagent le même espace d'observation O, $o^d \in O$ étant l'observation individuelle et partielle de l'agent d de son environnement environnant, agrégée à partir des observations de ses N_d nœuds. Par souci de clarté, il est important de noter que l'observation de chaque agent pour l'époque

κ inclut les besoins en ressources, la durée de vie et la localisation cartésienne de l'utilisateur pour le conteneur à déployer. Ces observations sont les mêmes pour tous les agents. De plus, l'observation de l'agent inclut les ressources disponibles et les emplacements cartésiens des nœuds N_d à l'époque κ.

À chaque intervalle, l'agent d choisit une action $a^d \in A^d$ qui correspond à sa décision de déployer ou non le conteneur demandé dans l'un de ses nœuds associés.

Par conséquent, le jeu a un espace d'action conjoint $\mathcal{A} = A^1 \times \ldots \times A^D$. Les agents perçoivent une récompense commune collaborative. Grâce au jeu de Markov, l'objectif est de maximiser la récompense actualisée définie comme $R_\kappa = \sum_{i=0}^{\infty} \gamma^i r_{\kappa+i}$, où r_κ est la fonction de récompense à l'époque κ.

Nous supposons ici que les nœuds $|\mathcal{F}|$ sont arbitrairement regroupés et associés à des agents d'apprentissage D. En pratique, le regroupement des nœuds peut être réalisé en fonction de la distance de communication entre les nœuds et les agents, de la disponibilité des ressources au sein des groupes formés, etc. De plus, un agent d'apprentissage peut être soit un appareil dédié, soit un nœud élu ayant une capacité de calcul suffisamment élevée pour gérer les requêtes des utilisateurs et les tâches de gestion des ressources.

Remarquez que dans le cas particulier où $D = |\mathcal{F}|$, chaque agent correspond à un seul nœud. Ainsi, ce dernier décide des demandes qu'il peut satisfaire, compte tenu uniquement de son état et de ses capacités.

6.4 Évaluation des performances

Dans cette section, nous décrivons les paramètres des simulations, puis les performances de nos solutions sont évaluées pour différents scénarios, et comparées aux solutions de l'état de l'art.

6.4.1 Paramètres des simulations

Environnement de simulation

Dans cette section, nous allons considérer le même cas d'utilisation de système routier intelligent déployé dans un environnement de *fog computing* dynamique, énoncé dans la section 4.4. Les paramètres de l'environnement de simulation sont résumés dans le Tableau 5.1. Nous définissons la capacité des ressources des nœuds comme décrit dans le Tableau 4.1 ainsi que les applications définit dans le Tableau 4.2. Dans notre système, les utilisateurs finaux uniformément distribués envoient des demandes de déploiement de conteneur au contrôleur de domaine du *fog* pour traiter les tâches de leurs applications, suivant une distribution de Poisson ayant un taux d'arrivée moyen λ.

Paramètres de l'apprentissage profond

L'architecture de l'algorithme collaboratif est composée d'un réseau de mixage et de plusieurs réseaux d'apprentissage (chacune pour un agent). Le réseau de mixage consiste en une seule couche cachée utilisant une fonction ReLu non-linéaire, alors que l'architecture de tous les réseaux d'agents est un réseau profond récurrente de la matrice Q (DRQN pour "Deep Recurrent Q-Network" en anglais) avec une couche composée d'unités récurrentes fermées (GRU) (Rashid *et al.*, 2018) . Nous avons aussi conduit des expérimentations a l'aide des outils Kubeflow et Katib pour déterminer l'architecture du réseau de neurone optimale à adopter. Cette recherche a été précisément faite dans le but de fixer le nombre de neurones dans chaque couche . Ce nombre étant compris 64 et 256 neurones. Enfin, la capacité de la mémoire de relecture, la taille du mini-ensemble de transitions et le facteur d'apprentissage ont été définis grâce au réglage des *hyper-paramètres* à l'aide des outils Kubeflow et Katib. La figure 6.4.1 illustre les réseaux employés par QMIX.

On voit dans le rectangle à gauche la structure du réseau de mixage. En rouge, les hyper-réseaux qui produisent les poids et les biais pour les couches de réseau de mélange indiquées en bleu. À droite, on voit la structure des réseaux des agents. Une meilleure visualisation en couleur.

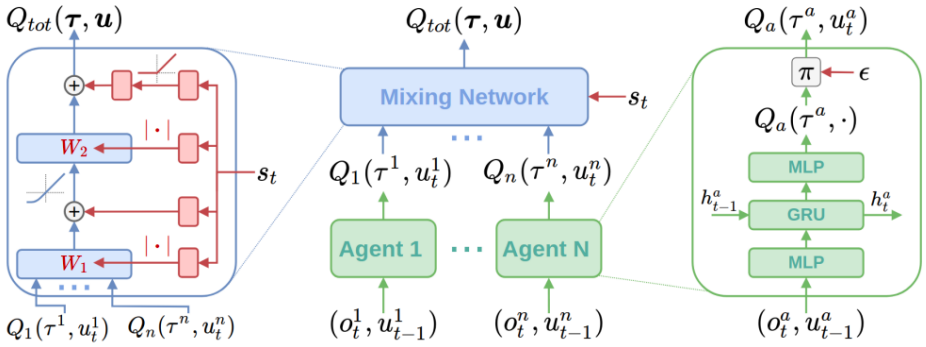


FIGURE 6.2 L'architecture globale de QMIX (Rashid *et al.*, 2018)

Sur la base des expériences menées, nous avons constaté que les meilleures performances pour l'architecture DNN centralisée en termes de convergence sont obtenues lors de l'utilisation d'un réseau de neurones à deux couches entièrement connecté, avec des couches cachées composées de 128 neurones ReLu. Outre la recherche de la meilleure architecture de réseau de neurones, nous avons également mené plusieurs expériences afin de déterminer certaines valeurs d'*hyper-paramètres*. Plus précisément, nous avons expérimenté plusieurs valeurs de capacité de mémoire tampon de relecture allant de 5000 à 10000, la taille du mini-ensemble de transitions comprise entre 128 et 512 transitions et un taux d'apprentissage allant de 10^{-6} à 10^{-4}. Le facteur actualisation a été initialisé à $\gamma = 0.9$.

6.4.2 Résultats des simulations

Nous présentons dans cette section l'effet de la variation de charge des utilisateurs, ainsi que l'effet de l'extensibilité du *fog*, de la mobilité des nœuds et du nombre

de nœuds par agent d'apprentissage sur les performances de notre algorithme collaboratif pour l'allocation des ressources. Nous supposons dans ces simulations qu'un agent gère 5 nœuds simultanément, sauf indication contraire. De plus, notre algorithme collaboratif proposé est comparé à l'algorithme d'allocation de ressources centralisé basé sur le DRL et aux méthodes suivantes de l'état de l'art :

- Attribution du nœud le plus proche de l'utilisateur (NF pour "Nearest Fog" en anglais) où le nœud disponible et géographiquement plus proche est sélectionné pour répondre à la demande de l'utilisateur ;
- Allocation d'un nœud aléatoire (RF pour "Random Fog" en anglais) où un nœud qui est disponible dans la portée de communication de l'utilisateur est attribué de manière aléatoire pour répondre à la demande de l'utilisateur ;
- l'algorithme VDN distribué (Sunehag *et al.*, 2018) ; et
- l'algorithme IQL distribué (Tan, 1993).

Les performances des algorithmes mentionnés ci-dessus sont comparées suivant différents scénarios avec les performances de la solution optimale du problème formulé en programme linéaire en nombre entier, dans la sous-section 5.2.1 et qui est obtenue à l'aide d'IBM CPLEX (CPLEX, 2016b).

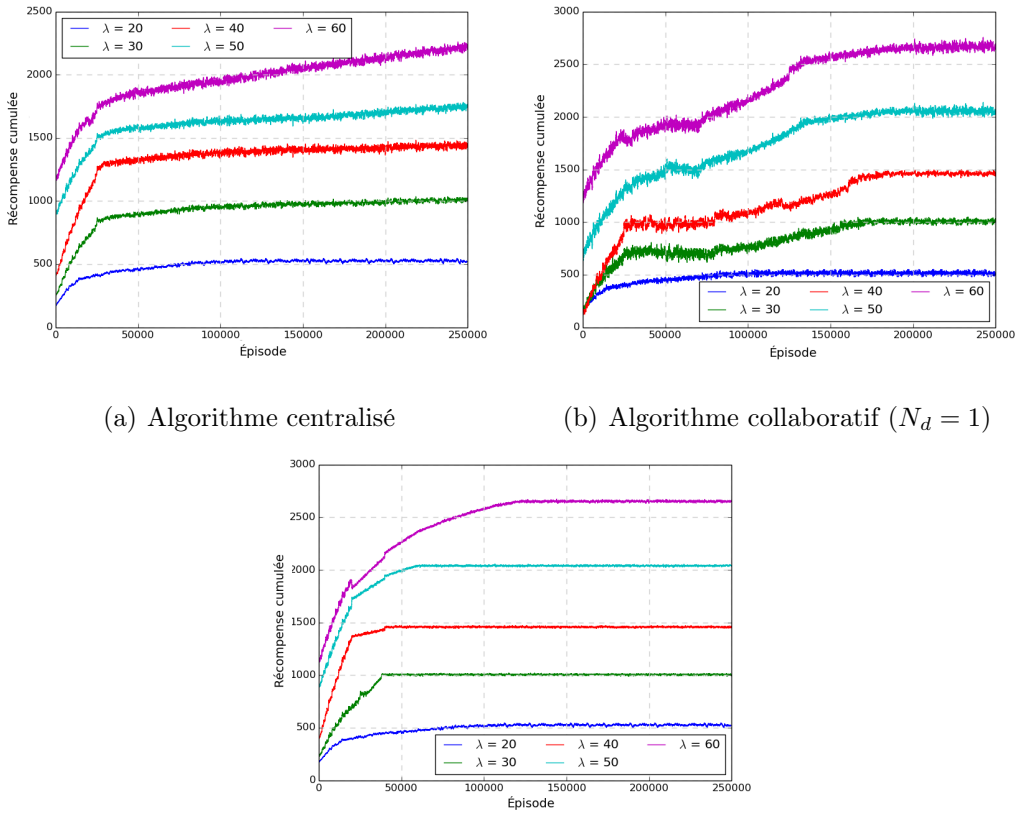
L'effet de la charge des requêtes utilisateurs

Dans ce paragraphe, nous étudions l'impact de λ sur la performance du taux de réussite. Ce taux est définit comme étant le ratio de toutes les requêtes répondues dans un délai inférieur au seuil prédéterminé (que nous appellerons aussi satisfaisantes) par rapport à toutes les requêtes reçu par le contrôleur. Pour notre algorithme, plus de 250K épisodes sont utilisés pour l'apprentissage, où chaque épisode est composé d'une configuration de 30 intervalles. Dans la Figure 6.3, les tendances d'apprentissage de nos algorithmes sont tracés à l'aide d'une fenêtre de moyenne de 100 épisodes.

Comme on peut le voir sur la Figure 6.3(a), l'algorithme centralisé converge vers sa politique optimale après 100K itérations pour $\lambda = 20$. Lorsque λ augmente, la convergence devient plus lente. Cela peut s'expliquer par le fait qu'un λ plus élevé (c'est-à-dire un nombre de requêtes plus élevé) génère un système dimensionnel plus élevé qui nécessite plus d'échantillons ou d'épisodes d'apprentissage. Nous pouvons aussi constater que pour $\lambda = 60$, l'algorithme centralise ne converge pas encore á une valeur optimale. Dans la Figure 6.3(b), l'algorithme collaboratif où chaque agent représente un unique nœud du *fog* ($N_d = 1$) converge en 100K épisodes pour $\lambda = 20$ et il a besoin d'un plus grand nombre d'épisodes lorsque λ augmente pour converger. Néanmoins, ses performances de convergence sont meilleures que celles de l'algorithme centralisé.

On note également que ces courbes sont obtenues en faisant la moyenne de 5 courbes de simulations et qu'avant convergence l'algorithme montre beaucoup de fluctuations qui s'expliquent par le fait que la convergence de l'algorithme basé sur les décisions d'un grand nombre d'agents prend du temps.

Enfin, la Fig. 6.3(c) illustre le comportement de convergence de l'algorithme collaboratif où chaque agent représente 5 nœuds du *fog* ($N_d = 5$), obtenus aussi en moyennant les résultats de 5 simulations. Cette approche permet d'obtenir une convergence plus rapide avec un nombre d'épisodes inférieur à 100K pour la plupart des valeurs de λ. En effet, cela peut s'expliquer par le nombre réduit des agents par rapport à la simulation précédente. Ceci va aussi engendrer des réseaux de neurones de plus petite taille que le réseau adopté dans l'algorithme centralisé. Par conséquent, l'algorithme collaboratif avec un nombre soigneusement sélectionné d'agents et une bonne stratégie de regroupement des nœuds pour un unique agent, devrait fournir les meilleures performances de convergence.



(a) Algorithme centralisé

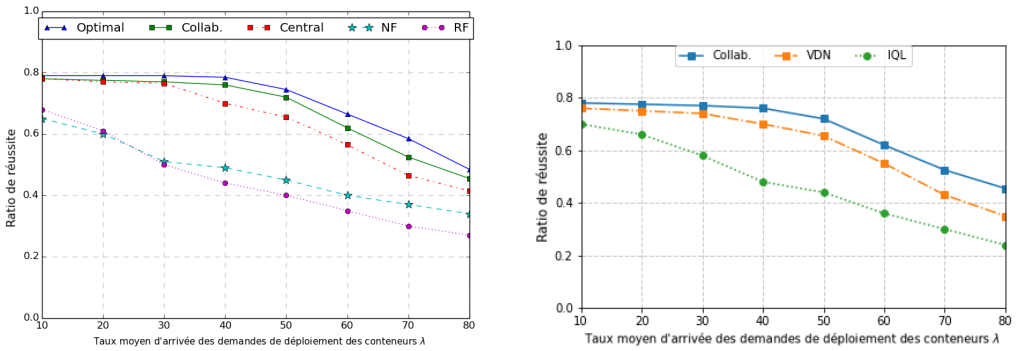
(b) Algorithme collaboratif ($N_d = 1$)

(c) Algorithme collaboratif ($N_d = 5$)

FIGURE 6.3 Convergence des algorithmes (différent taux moyen d'arrivée λ).

La Figure 6.4 illustre les performances des algorithmes proposés centralisés et collaboratifs ($N_d = 5$), des algorithmes de l'état de l'art et de la solution optimale. La Figure 6.4(a) montre que les algorithmes proposés atteignent des performances presque optimales avec une préférence pour l'approche collaborative pour les valeurs élevées de λ. Cela est principalement dû à la convergence rapide de l'approche collaborative par rapport à la méthode centralisée qui devrait nécessiter plus d'entraînement, comme le montre la Figure 6.3(a).

Il est aussi à noter que la résolution du problème (P1) à l'aide d'IBM CPLEX a pris entre 48 heures et 96 heures, selon la taille du scénario. En revanche, après



(a) Algorithmes collaboratif vs. centralisé (b) Algorithmes collaboratif vs. distribués

FIGURE 6.4 L'effet du taux moyen d'arrivée λ.

la phase d'entraînement, les résultats des approches centralisée et collaborative basées sur le renforcement profond ont été obtenus en moins d'une seconde. De plus, les deux algorithmes proposés surpassent les approches de l'état de l'art, en raison de leurs capacités d'apprentissage supérieures par rapport aux algorithmes gloutons tels que NF et RF.

La Figure 6.4(b) montre que les performances de l'algorithme collaboratif proposé surpasse les performances de l'algorithme basé sur l'approche VDN en termes de taux de requêtes satisfaisantes. Cela est dû au fait que notre algorithme réussi à estimer la fonction de valeur plus précisément que VDN. En revanche, IQL réalise les moins bonnes performances en raison de l'indépendance de l'apprentissage de ses agents. Plus précisément, chaque agent forme le réseau par lui-même, où il observe localement l'environnement pour prendre des décisions de son point de vue. Par conséquent, l'algorithme IQL est incapable d'apprendre une bonne politique en raison de la non-stationnarité de l'environnement causée en partie par le comportement variable des autres agents au cours de la formation. Néanmoins, dans la pratique, l'IQL est considéré comme une référence solide, pour les jeux mixtes et compétitifs (Rashid *et al.*, 2018; Tampuu *et al.*, 2017; Leibo *et al.*,

2017).

Effet du nombre de nœuds du *Fog*

Ces simulations étudient l'effet du nombre de nœuds sur les performances des algorithmes proposés. En raison de leurs complexité, les résultats optimaux ne peuvent être fournis que pour un maximum de 60 nœuds. Selon la Figure 6.5, l'algorithme collaboratif est sous-optimal et surpasse toutes les autres approches pour toutes les tailles de réseaux du *Fog*.

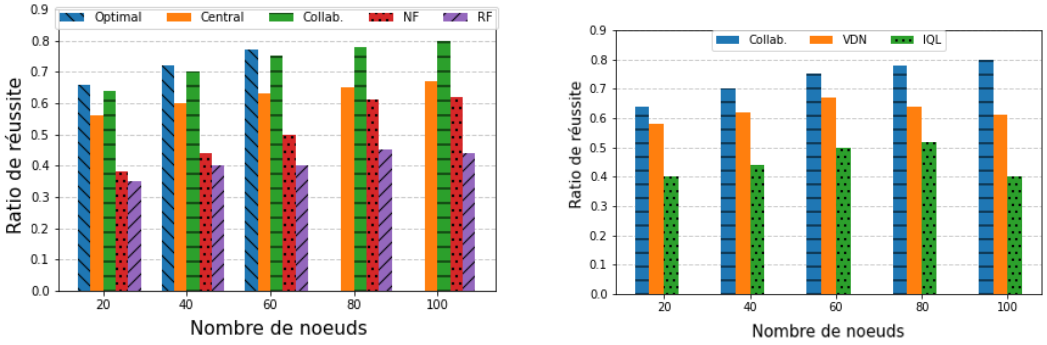
Selon la Figure 6.5(a), à mesure que $|\mathcal{F}|$ augmente, l'écart entre les algorithmes centralisé et collaboratif s'élargit. Cela est dû aux réseaux de neurones plus complexe et aux espaces d'actions et d'états plus grand pour l'approche centralisée ainsi qu'une phase d'entraînement qui n'a pas encore aboutie à la convergence de l'algorithme à sa valeur optimale au bout de 250K itérations. Ainsi, la solution est dite être moins extensible que la solution collaborative. On peut également constater que les algorithmes proposés surpassent ceux de l'état de l'art. Dans la Figure 6.5(b), nous remarquons que notre algorithme collaboratif surpasse en moyenne VDN de 10%, et cet écart augmente proportionnellement avec le nombre de nœuds. En effet, à mesure que le nombre de nœuds augmente, le VDN nécessite un réseaux de neurones plus complexe et un espace d'actions et d'états plus grand, dégradant ainsi ses capacités d'apprentissage et de convergence.

Effet de la mobilité des nœuds du *fog*

Nous évaluons les performances des algorithmes proposés et ceux de l'état de l'art pour différents scénarios de mobilité, où le taux d'arrivée des demandes est défini à $\lambda = 50$. Nous adoptons quatre scénarios de mobilité, à savoir :

— Nœuds immobiles (5 RSUs)

— Nœuds à mobilité lente (40 téléphones portables)



(a) Algorithmes centralisés vs. collaboratif (b) Algorithmes distribués vs. collaboratif

FIGURE 6.5 Effet du nombre de nœuds du *fog*

— Nœuds à mobilité rapide (20 véhicules)

— Nœuds à mobilité mixte (2 RSUs, 6 véhicules et 12 cellulaires)

La capacité totale des ressources disponibles dans les nœuds est la même pour tous les scénarios.

La convergence de l'algorithme collaboratif proposé est présentée pour les différents scénarios de mobilité dans la Figure 6.4.2. Le scénario où les nœuds sont immobiles a le même comportement que celui du scénario centralisé, tandis que pour les autres scénarios de mobilité, l'algorithme collaboratif atteint une convergence vers des politiques optimales plus rapide que celles de l'algorithme centralisé présenté dans la Figure 5.3 Par exemple, la récompense cumulée atteint sa valeur optimale de convergence dans les épisodes compris entre 55K et 100K pour les scénarios à mobilité lente et mixte, respectivement. De plus, les deux atteignent les mêmes performances que le scénario où les nœuds sont immobiles. En revanche, le scénario où les nœuds ont une mobilité rapide présente une convergence plus lente, vers une valeur de récompense cumulé plus faible. En effet, pour tous les scénarios de mobilité, la solution collaborative parvient à atteindre une convergence après au plus 150K épisodes alors que la solution centralisée n'a pas été en mesure d'atteindre cette convergence. Cela est principalement dû à la

plus petite taille des réseaux de neurones.

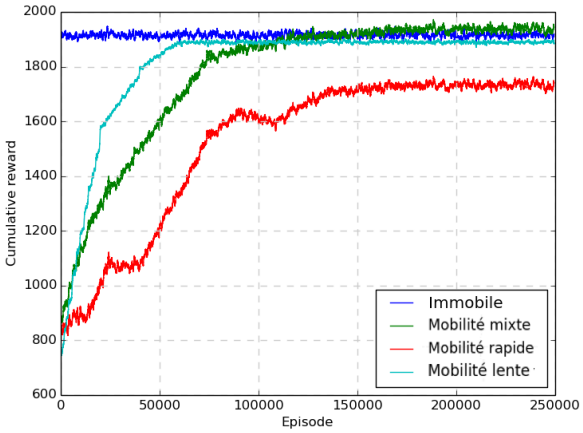
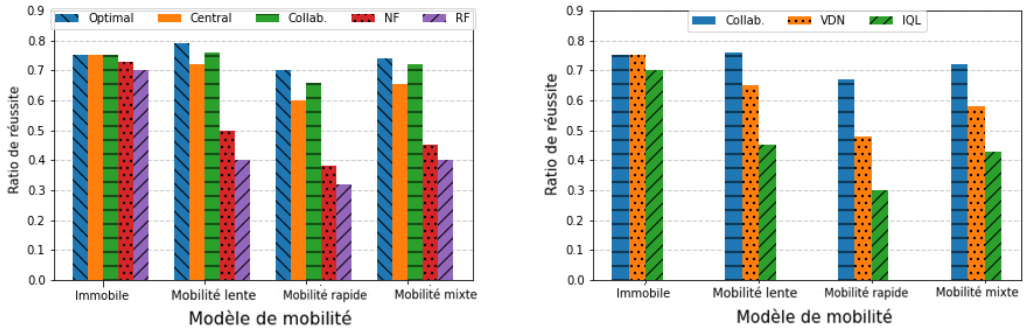


FIGURE 6.6 Convergence de l'algorithme collaboratif pour différents scénarios de mobilité

La Figure 6.7(a) compare les performances des algorithmes centralisés et collaboratif proposés par rapport aux performances de la solution optimale et des algorithmes NF et RF, pour différents scénarios de mobilité. Lorsque les nœuds sont immobiles, les algorithmes centralisés et collaboratif atteignent une performance optimale pour le taux de réussite, tandis que les algorithmes NF et RF atteignent des valeurs presque optimales. Cela peut s'expliquer par le nombre limité de nœuds qui rend l'apprentissage plus efficace pour ce scénario. Pour le scénario de mobilité lente, l'algorithme collaboratif atteint des performances quasi optimales, tandis que l'algorithme centralisé réalise un taux de réussite est inférieur de 5%, et que les algorithmes NF et RF présentent respectivement des taux de réussite inférieurs de 55% et de 45%. Dans le scénario de mobilité rapide, les performances de tous les algorithmes se dégradent davantage, avec une meilleure performance pour l'approche collaborative proposée. Enfin, étant donné les schémas de mobilité des nœuds mixtes, l'algorithme collaboratif atteint un taux de réussite quasi optimal qui surpasse l'algorithme centralisé de 10%, et les

algorithmes NF et RF de plus de 30%.

Comme pour la Figure précédente, la Figure 6.7(b) évalue les performances de l'algorithme collaboratif proposé par rapport aux performances de la solution optimale et des algorithmes VDN et IQL, compte tenu de différents scénarios de mobilité. Lorsque les nœuds sont immobiles, la solution collaborative et le VDN atteignent des performances optimales, tandis que IQL atteint des performances légèrement inférieures. Cela peut s'expliquer par la stationnarité de l'environnement. Pour les autres scénarios de mobilité, les performances des algorithmes VDN et IQL sont nettement inférieures à ceux de la solution collaborative proposée. Par exemple, pour le scénario de mobilité rapide, seulement 30% et 43% des demandes ont été satisfaites respectivement pour IQL et VDN, contre 68% pour notre approche. Ces résultats démontrent la robustesse de notre solution dans des environnements très dynamiques. En effet, la forte mobilité et les changements d'environnement ont ralenti la convergence des autres solutions.



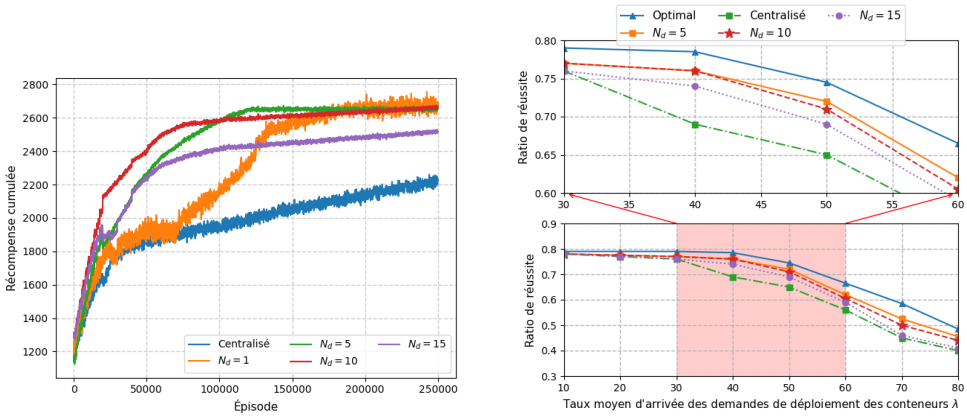
(a) Algorithmes collaboratifs vs centralisés (b) Algorithmes collaboratifs vs. distribués

FIGURE 6.7 Effect de la mobilité des nœuds

Effet du nombre de nœuds du *fog* par agent d'apprentissage

Dans notre modèle d'apprentissage collaboratif, chaque agent d'apprentissage dispose d'une vue partielle de l'état de l'environnement. Plus précisément, il ne connaît que l'état d'un ensemble prédéfini de nœuds N_d et utilise ces informations pour décider ces nœuds peuvent satisfaire ou non les demandes entrantes, en fonction de l'état de mobilité, de disponibilité et de charge de ses N_d nœuds . Dans ces simulations, nous considérons que cet agent est un nœud choisi aléatoirement parmi un groupe de $N_d + 1$ nœuds. Dans ces simulations, nous considérons quatre scénarios distincts où $N_d \in \{1, 5, 10, 15\}$. Dans le cas particulier de $N_d = 1$, un agent correspond à un seul nœud. De plus, nous considérons un modèle de mobilité mixte de 60 nœuds et un taux d'arrivée moyen des demandes $\lambda = 60$.

La Figure 6.8(a) compare la convergence de l'algorithme collaboratif à celle de l'algorithme centralisé pour différentes valeurs N_d, . Dans tous les scénarios, la méthode collaborative offre une meilleure récompense cumulative que la méthode centralisée. En effet, répartir les décisions sur différents agents de RL ayant une observation partielle de l'environnement accélère le processus d'apprentissage. Pour le scénario $N_d = 1$ où chaque nœud du Fo représente un agent, la récompense cumulée converge lentement (et n'a toujours pas une convergence optimale après 250K épisodes). Lorsque $N_d = 10$ et $N_d = 15$, la convergence des algorithmes suit une même tendance rapide, mais avec une meilleure récompense cumulée pour $N_d = 10$. En revanche, lorsque $N_d = 5$, l'algorithme collaboratif converge après 120K épisodes et obtient la meilleure récompense cumulée. À la lumière de ces résultats, on remarque qu'il y a un compromis à faire entre le nombre d'agents RL (N_d) et la vitesse de convergence. Plus précisément, lorsque N_d est petit, un nombre élevé d'agents RL est déployé. Ces agents prennent des décisions qui affectent implicitement l'apprentissage de l'autre via la valeur de récompense, ce



(a) Convergence des algorithmes suivant différentes valeurs de N_d

(b) Taux de succés suivant plusieurs valeurs de N_d

FIGURE 6.8 Effet du nombre de nœuds du Fog par agent d'apprentissage

qui ralentit la convergence de l'algorithme. En revanche, un plus petit nombre d'agents RL signifie moins d'influence sur l'apprentissage les uns des autres, accélérant ainsi la convergence de l'algorithme. Par la suite, la valeur N_d doit être soigneusement optimisée pour déterminer la meilleure stratégie distribuée des agents RL.

La Figure 6.8(b) montre l'effet de N_d sur le taux de réussite pour différents taux d'arrivée de requêtes λ. Pour toutes les valeurs de N_d, le taux de réussite de l'algorithme collaboratif est supérieur à celui de l'algorithme centralisé pour toutes les valeurs de λ. Le scénario avec $N_d = 5$ atteint la meilleure performance globale de taux de réussite, qui est la plus proche du résultat optimal. Autrement dit, $N_d = 5$ représente la meilleure stratégie de distribution des agents RL.

6.5 Conclusion

Dans ce chapitre, nous avons proposé une méthode basée sur QMIX pour l'allocation de ressources pour un environnement de FC dynamique. Puis, nous

avons évalué les performances de cette approche et les avons comparées à d'autres approches de l'état de l'art. Les résultats obtenus illustrent la quasi-optimalité atteinte par l'approche collaborative proposée, et sa supériorité par rapport aux méthodes conventionnelles en termes de taux de réussite. En outre, nous avons constaté que l'algorithme centralisé est pratique pour les systèmes à petite échelle, tandis que l'algorithme collaboratif est plus adéquat pour les systèmes à plus grande échelle. Enfin, l'étude de l'impact des paramètres du système, comme le taux d'arrivée des requêtes, le nombre de nœuds et la mobilité sur les performances en terme de taux de succès a illustré la robustesse de la solution collaborative proposée.

Algorithme 4 Algorithme collaboratif basé sur le RL

Initialiser la mémoire de relecture $\mathcal{D}$ et la fonction action-valeur Q_{tot} avec des poids aléatoires
pour agent $d = 1 \dots D$ **faire**
    Initialiser la mémoire de relecture $\mathcal{D}^d$ et la fonction action-valeur Q^d avec des poids aléatoires
fin pour
pour episode $= 1 \dots M$ **faire**
    Initialiser la séquence s_1
    pour $\kappa = 1 \dots K$ **faire**
        pour agent $d = 1 \dots D$ **faire**
            Définir l'observation $o_\kappa^d = \{\mathbf{c}_R^\kappa, \mathbf{c}_L^\kappa, G_\kappa, \mathbf{F}_R^{\kappa,d}, \mathbf{F}_L^{\kappa,d}\}$
            Suivant une probabilité ϵ sélectionner une action aléatoire $a_\kappa^d \in A^d$, sinon sélectionner $a_\kappa^d = \max_{a^d} Q_d^*(o_\kappa^d, a^d; \theta)$
        fin pour
        Exécuter l'action a_κ and observer la récompense r_κ, comme défini dans l'Equation (5.8), le nouvel état $s_{\kappa+1}$ et l'ensemble des observations $O_{\kappa+1}$
        pour agent $d = 1 \dots D$ **faire**
            Garder la transition $e_\kappa^d = (o_\kappa^d, a_\kappa^d, r_\kappa, o_{\kappa+1}^d)$ dans $\mathcal{D}^d$
            Extraire un petit ensemble aléatoire de transitions $(o_j^d, a_j^d, r_j, o_{j+1}^d)$ de $\mathcal{D}^d$ et entraîner les réseaux des agents
            $y_j^d = \begin{cases} r_j, \text{Si l'etat } o_{j+1} \text{ est final} \\ r_j + \gamma \max_{a^d} Q^d(o_{j+1}^d, a^d; \theta_d^-), \text{Sinon.} \end{cases}$
            Effectuer une descente de gradient avec la perte $L(\theta_d)$ (eq. (5.6))
            Mettre à jour le vecteur de poids θ_d^- chaque τ étapes
        fin pour
        Garder la transition $e_\kappa = (s_\kappa, a_\kappa, r_\kappa, s_{\kappa+1})$ dans $\mathcal{D}$
        Extraire un petit ensemble aléatoire de transitions (s_j, a_j, r_j, s_{j+1}) de $\mathcal{D}$ et entraîner les réseaux des agents
        $y_j = \begin{cases} r_j, \text{Si l'etat } s_{j+1} \text{ est terminal} \\ rj + \gamma \max_a Q(s_{j+1}, a; \theta^-), \text{Sinon.} \end{cases}$
        Effectuer une descente de gradient avec la perte $L(\theta)$ (Eq. (5.6))
        Mettre à jour le vecteur de poids θ^- chaque τ étapes
    fin pour
fin pour

CHAPITRE VII

CONCLUSION ET PERSPECTIVES

Ce chapitre fournit une synthèse des travaux présentés dans cette thèse, présente les travaux en cours qui complètent cette thèse et fournit de nouvelles pistes de recherche pour les travaux futurs.

7.1 Synthèse des travaux

Nous avons présenté tout au long de cette thèse des algorithmes pour l'allocation des ressources dans les environnements de *fog computing* dynamiques. En effet, les nœuds du *fog* sont hétérogènes, géographiquement distribués, mobiles, extensibles, et la disponibilité de leurs ressources est sporadique. D'autre part, les utilisateurs qui communiquent avec le *fog* sont aussi hétérogènes, et la charge de leurs requêtes est variable. Or, une allocation qui ne prend pas en compte ce dynamisme influencerait d'une manière négative les performances des applications. Dans cette thèse, nous avons proposé des stratégies pour l'allocation des ressources pour cette infrastructure de *fog* dynamique. En outre, ces stratégies prennent en considération l'aspect dynamique, largement distribué et très extensible des nœuds du *fog*. Les contrôleurs des différents domaines du *fog* exécutant ces stratégies visent à maximiser le nombre de requêtes dont le délai de réponse ne dépasse pas un seuil prédéterminé tout en équilibrant la charge des différents nœuds et en minimisant les migrations des applications entre les nœuds en temps réel.

Nous avons, premièrement, étudié les différents travaux de l'état de l'art qui se sont intéressés aux problèmes d'allocation de ressources et de provisionnement des tâches dans les environnements de *fog computing*. Nous avons soulevé les limitations et les faiblesses des travaux qui ont proposé des solutions *offlines* et *onlines*, centralisées et distribuées. En effet, un bon nombre de ces travaux a négligé ou partiellement considéré les effets des comportements dynamiques des nœuds du *fog*. Nous avons aussi étudié les métriques de performances adoptées dans ces travaux. En outre, le délai de réponse était une métrique largement utilisée ou remplacée par une contrainte relative au délai.

Ensuite, nous avons présenté l'architecture de *fog computing* considérée dans notre travail, ainsi que les différents modules qui la composent. Nous avons formulé comme un programme linéaire en nombres entiers le problème d'allocation des ressources étudié. Cette formulation permet de fournir un outil de résolution efficace pour obtenir la solution optimale en utilisant des boîtes à outils standards. Ces résultats seront considérés comme des références de comparaison et d'évaluation de performances.

Par la suite, nous avons proposé trois méthodes pour résoudre les problèmes d'allocation des ressources. D'abord, la méthode *offline* centralisée, où l'environnement du *fog* est divisé en domaines. Chaque domaine englobe un ensemble de nœuds et un contrôleur qui peut prendre les décisions d'allocation des ressources en ayant toutes les entrées nécessaires. En outre, nous avons étudié conjointement les problèmes de placement des conteneurs et le problème de provisionnement des tâches pour un environnement dynamique de *fog computing*. La mobilité des nœuds et la variation sporadique de la charge des fonctions principales ont été prises en compte. Nous avons proposé une méta-heuristique basée sur l'Optimisation par essaims particulaires et une heuristique gloutonne pour résoudre le problème d'optimisation étudié. Ces solutions sont évaluées par

des simulations sur un système routier intelligent déployé dans un environnement de *fog computing* dynamique et en utilisant des données réelles de mobilité. Nous avons montré que l'algorithme basé sur le OEP obtient des résultats quasi-optimaux, mais avec des temps d'exécution plus longs que l'algorithme glouton. En effet, l'algorithme glouton obtient des résultats inférieurs de 10% à 30% par rapport aux résultats optimaux avec un temps d'exécution négligeable. Enfin, nous avons étudié l'impact de la variation de la charge des fonctions de base. Cependant, la variation des ressources dans les nœuds du *fog*, due à la charge variable des fonctions de base, peut dégrader significativement les performances du système.

Ces algorithmes peu-complexes devraient aboutir à des solutions tout en ayant les paramètres complets du problème comme entrée. La qualité de ces solutions est garantie théoriquement, mais ne représente pas une solution réaliste étant donné que les paramètres du problème ne peuvent pas être connus depuis le début. Pour remédier à ceci, nous proposerons des algorithmes *onlines*, centralisés et distribués basés sur l'apprentissage machine permettant d'allouer les ressources dans les différents nœuds en tenant compte de leur aspect dynamique, extensible et largement distribué.

Notre deuxième contribution a donc consisté à proposer une solution *online* centralisée. D'abord, nous avons formulé le problème de placement de conteneurs dans les nœuds du *fog* en vue de maximiser le nombre de requêtes dont le délai de réponse est inférieur à un seuil sous la forme d'un programme linéaire en nombres entiers. Puis, nous avons prouvé sa NP-Difficulté. Ensuite, nous avons proposé une solution qui repose sur l'apprentissage par renforcement pour résoudre d'une manière en ligne le problème étudié. À chaque intervalle de temps, le contrôleur de domaine reçoit une nouvelle demande de placement de conteneur, et, devra décider sur la base des paramètres appris le noœud du *fog* qui va héberger le

conteneur. L'algorithme *online* a aussi pour objectif de placer les conteneurs de manière à servir les utilisateurs finaux dans la limite du délai prédéterminé. Les performances de cet algorithme sont évaluées par des simulations utilisant des données de mobilité réelles. Les résultats montrent que notre solution atteint des résultats quasi-optimaux en termes de taux de satisfaction, nettement meilleurs que les approches de l'état de l'art. Cependant, cette solution ne répond pas adéquatement à toutes les exigences du *fog* car elle ne tient pas compte de certaines caractéristiques. En effet, les simulations effectuées sur un nombre élevé de nœuds ne présentaient pas des résultats de convergences optimaux. Ceci s'explique par la taille élevée des réseaux de neurones résultants de ces environnements. Ce qui diminue la convergence de ces algorithmes vers des solutions optimales et dégrade ainsi les performances de l'algorithme. Pour cette raison, nous avons proposé une solution *online* distribuée.

La troisième contribution de cette thèse est la méthode *online* distribuée. Dans cette méthode, nous considérons plusieurs agents d'apprentissage dans un même domaine. Chaque agent, observe uniquement le comportement d'un groupe de nœuds et décide si l'un de ses nœuds devrait héberger le conteneur demandé ou non. Cette contribution est différente de la précédente, étant donné que les agents collaborent ensemble pour décider du nœud qui va héberger le conteneur. Leur apprentissage est basé sur leurs observations locales relatives à un ensemble de nœuds qui leurs sont dédiés et aux récompenses reçues suite à leur décision collective. Pour ce faire, nous avons adopté la méthode QMIX pour l'apprentissage collaboratif distribué. L'architecture de QMIX se compose de l'ensemble de réseaux des agents, et d'un réseau de mixage qui les combine, non pas comme une simple somme, mais d'une manière non linéaire complexe qui assure la cohérence entre la politique centralisée et celle décentralisée. Les performances de cette approche, comparées à d'autres approches de l'état de l'art,

illustrent la quasi-optimalité atteinte et sa supériorité par rapport aux méthodes conventionnelles en termes de taux de réussite. En outre, nous avons constaté que l'algorithme centralisé est pratique pour les systèmes à petite échelle, tandis que l'algorithme collaboratif est plus adéquat pour les systèmes à plus grande échelle.

7.2 Contribution à l'avancement des connaissances

Cette thèse vise à contribuer à l'avancement des connaissances dans le domaine du *fog computing* grâce à la conception et l'évaluation d'algorithmes d'allocation des ressources pour des applications sensibles à la latence. Ce type d'infrastructures constitue sans doute la clé de voûte de la technologie de l'IoT et les applications de réalité virtuelle futures. Spécifiquement, ce projet de doctorat, grâce aux solutions spécifiquement conçues, aura une incidence sur les applications en temps réel, nécessitant d'être déployé à la périphérie du réseau. Ces résultats, à la fois originaux et efficaces, apportent des solutions et des bases solides pour propulser le développement des applications temps réel, notamment à l'ère de l'émergence de l'IoT et de l'intelligence artificielle. Plus particulièrement, nous souhaitons que cette thèse ait une incidence sur :

- une meilleure compréhension des infrastructures de *fog computing*. Ceci implique la nature des nœuds et leurs comportements, ainsi que ceux des utilisateurs finaux.
- le développement des algorithmes pour l'allocation des ressources dans les infrastructures se trouvant à la périphérie du réseau se basant sur des nœuds périphériques à comportement dynamique.
- le déploiement d'orchestrateurs pour ces infrastructures de *fog computing* qui prennent en compte les spécificités de ces environnements.

Tous les résultats proposés tout au long de cette thèse sont, au meilleur de nos connaissances, à la fois nouveaux et uniques en termes d'efficacité et

de praticabilité. Par conséquent, ils constituent des innovations technologiques importantes.

7.3 Travaux en cours

La collaboration entre les différents domaines du *fog* devrait permettre une optimisation du partage des ressources. En effet, dans (Souza *et al.*, 2020) les auteurs mettent en évidence que la coopération entre les différents domaines permet de maximiser la disponibilité des ressources dans les domaines du *fog* en minimisant la fréquence de transfert des données et des migrations de conteneurs, en particulier pour les environnements mobiles dynamiques. Pour permettre cette collaboration, les agents doivent avoir certaines règles prédéfinies qui régissent l'échange des informations entre eux (Wang *et al.*, 2018a). De ce fait, deux exigences essentielles doivent être satisfaites : a) la mise en œuvre des standards inter-domaines et d'interfaces opérationnelles et b) le développement de plateformes de gestion de la mobilité et de fédération efficaces (Habibi *et al.*, 2020). Par ailleurs ces exigences sont largement ignorées dans les solutions actuelles et doivent être prises en compte dans les recherches futures.

Nous nous sommes basés sur l'architecture de référence de l'organisme ETSI pour l'informatique multi-accès à la périphérie (MEC pour "Multi-access Edge Computing" en anglais) (Etsi, 2019). Cette architecture décrit un système de périphérie, qui permet aux applications de s'exécuter de manière efficace et transparente dans un réseau multi-accès. Cette architecture décrit également les éléments fonctionnels et les points de référence entre eux, ainsi qu'un certain nombre de services MEC qui composent la solution. Sur cette référence de base, nous avons conçu une proposition d'architecture inter-domaines où une orchestration de ressources inter-domaines pouvait être possible. Cet orchestrateur pourrait éventuellement implémenter l'algorithme d'allocation des ressources

collaboratif proposé dans cette thèse. Ce travail est encore en cours de réalisation, nous sommes encore à l'étape de l'implémentation de la solution, que nous devrions par la suite valider.

7.4 Perspectives

Les travaux réalisés dans cette thèse ouvrent plusieurs pistes de recherche pour le futur. Parmi ces pistes nous citons les suivantes.

— Cette thèse considère souvent la fonction objectif qui maximise le nombre d'utilisateurs servis dans un délai supérieur au seuil prédéfini. L'étude d'autres fonctions objectif est évidemment une piste de recherche importante et ouverte. Évidemment, l'extension des algorithmes proposés dans cette thèse pour qu'ils optimisent d'autres fonctions objectif est également importante.

— Dans le chapitre 6 de cette thèse, pour permettre la prise de décision collaborative, un nœud est responsable d'un groupe de nœuds choisis aléatoirement. Cependant la sélection des nœuds physiques de contrôle et la délimitation des domaines sous leurs responsabilités devrait être considérées. Ce choix influence les délais de contrôle et donc les délais de réponse aux requêtes des utilisateurs. Dans un système de *fog computing*, un contrôleur gère un ensemble de nœuds physiques. En outre, la délimitation des domaines de contrôle va influencer le processus d'allocation de ressources. Un contrôleur va allouer les ressources dans les nœuds qu'il contrôle d'abord afin de diminuer le temps requis par le processus d'allocation des ressources. Modéliser une architecture de contrôleurs hybride (hiérarchique centralisé et pair à pair distribué) et la comparer aux solutions hiérarchiques et totalement distribués proposées dans la littérature.

— Dans cette thèse, nous avons uniquement considéré l'aspect d'allocation de ressources pour les environnements de *fog* dynamiques. Cependant, d'autres aspects, tels que la découverte de nœuds ou l'ordonnancement des tâches devraient être spécifiquement étudiés dans les environnements dynamiques.

BIBLIOGRAPHIE

Akjiratikarl, C., Yenradee, P. et Drake, P. R. (2007). Pso-based algorithm for home care worker scheduling in the uk. *Computers & Indus. Engineering*, *53*(4), 559–583.

Al-Khafajiy, M., Baker, T., Waraich, A., Al-Jumeily, D. et Hussain, A. (2018). Iot-fog optimal workload via fog offloading. Dans *Int. IEEE/ACM Conf. on Utility and Cloud Comput. Companion (UCC Companion)*, 359–364.

Alam, M. G. R., Tun, Y. K. et Hong, C. S. (2016). Multi-agent and reinforcement learning based code offloading in mobile fog. Dans *Proc. Int. Conf. Info. Net. (ICOIN)*, 285–290.

AMPL (2016). *AMPL : Streamlined Modeling for Real Optimization*. Récupéré de `http://ampl.com`

Araniti, G., Campolo, C., Condoluci, M., Iera, A. et Molinaro, A. (2013). LTE for Vehicular Networking : A Survey. *IEEE Commun. Mag.*, *51*(5), 148–157.

Baburao, D., Pavankumar, T. et Prabhu, C. (2021). Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method. *Applied Nanoscience*, 1–10.

Baker, T., Ugljanin, E., Faci, N., Sellami, M., Maamar, Z. et Kajan, E. (2018). Everything as a Resource : Foundations and Illustration through Internet-of-Things. *Computers in Industry*, *94*, 62–74.

Betke, M., Haritaoglu, E. et Davis, L. S. (2000). Real-time Multiple Vehicle Detection and Tracking from a Moving Vehicle. *Machine Vision and Appl.*, *12*(2), 69–83.

Bitsakos, C., Konstantinou, I. et Koziris, N. (2018). DERP : A deep reinforcement learning cloud system for elastic resource provisioning. Dans *Proc. IEEE Int. Conf. Cloud Comput. Tech. and Sci. (CloudCom)*, 21–29.

Bracciale, L., Bonola, M., Loreti, P., Bianchi, G., Amici, R. et Rabuffi, A. (2014). CRAWDAD Dataset Roma/Taxi. Downloaded from `https://crawdad.org/roma/taxi/20140717`.

Busoniu, L., Babuska, R. et De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. on Systems, Man, and Cybernetics, Part C (Appl. and Reviews)*, *38*(2), 156–172.

C. Arth, F. L. et Bischof, H. (2007). Real-Time License Plate Recognition on an Embedded DSP-Platform. Dans *Proc. IEEE Conf. on Comput. Vision and Pattern Recogn.*, 1–8. http://dx.doi.org/10.1109/CVPR.2007.383412

Chen, T., Ling, Q., Shen, Y. et Giannakis, G. B. (2018). Heterogeneous online learning for "Thing-Adaptive" fog computing in IoT. *IEEE IoT J.*, *5*(6), 4328–4341. http://dx.doi.org/10.1109/JIOT.2018.2860281

Chen, X., Leng, S., Zhang, K. et Xiong, K. (2019). A machine-learning based time constrained resource allocation scheme for vehicular fog computing. *China Commun.*, *16*(11), 29–41.

Chen, X., Pu, L., Gao, L., Wu, W. et Wu, D. (2017). Exploiting Massive D2D Collaboration for Energy-Efficient Mobile Edge Computing. *IEEE Wireless Commun.*, *24*(4), 64–71. http://dx.doi.org/10.1109/MWC.2017.1600321

Cheng, Z., Min, M., Liwang, M., Huang, L. et Gao, Z. (2022). Multiagent ddpg-based joint task partitioning and power control in fog computing networks. *IEEE IoT J.*, *9*(1), 104–116.

Cisco (2015). Cisco fog computing solutions : Unleash the power of the Internet of things. Dans *White Paper*. Récupéré de http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-solutions.pdf

Clerc, M. et Kennedy, J. (2002). The Particle Swarm-explosion, Stability, and Convergence in a Multidimensional Complex Space. *IEEE Trans. Evol. Comput.*, *6*(1), 58–73.

Coelho, L. C. (2013). *Linearization of the Product of Two Variables*. Récupéré de https://www.leandro-coelho.com/linearization-product-variables/

CPLEX (2016a). *Branch and cut in CPLEX*. Récupéré de https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.2/ilog.odms.cplex.help/refcppcplex/html/branch.html

CPLEX (2016b). *IBM CPLEX Optimizer*. Récupéré de http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Cui, D., Peng, Z., Jianbin, X., Xu, B. et Lin, W. (2018). A reinforcement learning-based mixed job scheduler scheme for grid or IaaS cloud. *IEEE Trans. Cloud Comput.*, *8*(4), 1030–1039.

De Brito, M. S., Hoque, S., Magedanz, T., Steinke, R., Willner, A., Nehls, D., Keils, O. et Schreiner, F. (2017). A service orchestration architecture for fog-enabled infrastructures. Dans *2017 Second Int. Conf. on Fog and Mobile Edge Computing (FMEC)*, 127–132. IEEE.

Deng, R., Lu, R., Lai, C., Luan, T. H. et Liang, H. (2016). Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption. *IEEE Internet of Things J.*, *3*(6), 1171–1181.

Eberhart, R. C. et Shi, Y. (1998). Comparison Between Genetic Algorithms and Particle Swarm Optimization. Dans *Proc. Int. Conf. on Evolutionary Program.*, 611–616. Springer.

Etsi, M. (2019). Multi-access edge computing (mec) ; framework and reference architecture. *ETSI GS MEC*, *3*, V2.

Fan, Q., Bai, J., Zhang, H., Yi, Y. et Liu, L. (2022). Delay-aware resource allocation in fog-assisted iot networks through reinforcement learning. *IEEE Internet of Things Journal*, *9*(7), 5189–5199. `http://dx.doi.org/10.1109/JIOT.2021.3111079`

Fang, C., Xu, H., Yang, Y., Hu, Z., Tu, S. S., Ota, K., Yang, Z., Dong, M., Han, Z., Yu, F. R. et Liu, Y. (2022). Deep reinforcement learning based resource allocation for content distribution in fog radio access networks. *IEEE IoT J.*, 1–1.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N. et Whiteson, S. (2018). Counterfactual multi-agent policy gradients. Dans *Proc. of the AAAI Conf. on artificial intelligence*, volume 32.

Fogel, D. B. (2006). *Evolutionary computation : toward a new philosophy of machine intelligence*. John Wiley & Sons.

Gao, X., Huang, X., Bian, S., Shao, Z. et Yang, Y. (2020). PORA : Predictive offloading and resource allocation in dynamic fog computing systems. *IEEE IoT J.*, *7*(1), 72–87.

Garey, M. R. et Johnson, D. S. (2002). *Computers and Intractability*, volume 29. WH Freeman New York.

Golberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, *1989*(102), 36.

Group, O. C. A. W. (2017). Openfog reference architecture for fog computing. Récupéré de `http://www.openfogconsortium.org/resources/`

white-papers/

Gupta, P. et Kumar, P. R. (2000). The Capacity of Wireless Networks. *IEEE Trans. Inf. Theory*, *46*(2), 388–404.

Habibi, P., Farhoudi, M., Kazemian, S., Khorsandi, S. et Leon-Garcia, A. (2020). Fog computing : a comprehensive architectural survey. *IEEE Access*, *8*, 69105–69133.

Haghi Kashani, M., Rahmani, A. M. et Jafari Navimipour, N. (2020). Quality of service-aware approaches in fog computing. *International Journal of Communication Systems*, *33*(8), e4340.

Helsley, M. (2009). LXC : Linux container tools. *IBM DeveloperWorks Technical Library*, *11*.

Hoang, D. et Dang, T. D. (2017). FBRC : Optimization of Task Scheduling in Fog-based Region and Cloud. Dans *IEEE Trustcom/BigDataSE/ICESS*, 1109–1114.

Hou, X., Li, Y., Chen, M., Wu, D., Jin, D. et Chen, S. (2016). Vehicular Fog Computing : A Viewpoint of Vehicles as the Infrastructures. *IEEE Trans. Veh. Tech.*, *65*(6), 3860–3873. http://dx.doi.org/10.1109/TVT.2016.2532863

Hu, J., Wellman, M. P. *et al.* (1998). Multiagent reinforcement learning : theoretical framework and an algorithm. Dans *ICML*, volume 98, 242–250. Citeseer.

IEEE Std. (2009). *802.11n-2009 : Enhancements for higher throughput*. Récupéré de http://www.ieee802.org

Injong, R., Minsu, S., Seongik, H., Seongjoon, L. K. K. et Song, C. (2009). CRAWDAD Dataset Ncsu/Mobilitymodels. Downloaded from https://crawdad.org/ncsu/mobilitymodels/20090723.

Jalali, F., Hinton, K., Ayre, R., Alpcan, T. et Tucker, R. S. (2016). Fog computing may help to save energy in cloud computing. *IEEE Journal on Selected Areas in Commun.*, *34*(5), 1728–1739.

Jamil, B., Ijaz, H., Shojafar, M., Munir, K. et Buyya, R. (2022). Resource allocation and task scheduling in fog computing and internet of everything environments : A taxonomy, review, and future directions. *ACM Computing Surv. (CSUR)*.

Jiang, Y., Huang, Z. et Tsang, D. H. (2017). Challenges and solutions in fog computing orchestration. *IEEE Netw.*

Jiang, Y., Huang, Z. et Tsang, D. H. K. (2018). Challenges and solutions in fog computing orchestration. *IEEE Netw.*, *32*(3), 122–129.

Jiao, L., Friedman, R., Fu, X., Secci, S., Smoreda, Z. et Tschofenig, H. (2013). Cloud-based computation offloading for mobile devices : State of the art, challenges and opportunities. Dans *Proc. Future Network Mobile Summit*, 1–11.

Kennedy, J. et Eberhart, R. (1995). Particle Swarm Optimization. Dans *Proc. IEEE Int. Conf. Neural Net.*, volume 4, 1942–1948.

Kubeflow. (2020a). *Introduction to Katib*. Récupéré de `https://www.kubeflow.org/docs/components/katib/overview/`

Kubeflow. (2020b). *Kubeflow*. Récupéré de `{https://www.kubeflow.org/}`

kubernetes (2018). *Kubernetes : Assign CPU Resources to Containers and Pods*. Récupéré de `https://kubernetes.io/docs/tasks/configure-pod-container/assign-cpu-resource/`

Lee, G., Saad, W. et Bennis, M. (2019). An online optimization framework for distributed fog network formation with minimal latency. *IEEE Trans. Wireless Commun.*, *18*(4), 2244–2258. `http://dx.doi.org/10.1109/TWC.2019.2901850`

Lee, S. et Lee, S. (2020). Resource allocation for vehicular fog computing using reinforcement learning combined with heuristic information. *IEEE IoT J.*, *7*(10), 10450–10464.

Leibo, J. Z., Zambaldi, V., Lanctot, M., Marecki, J. et Graepel, T. (2017). Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv :1702.03037*.

Liu, F., Shu, P., Jin, H., Ding, L., Yu, J., Niu, D. et Li, B. (2013). Gearing resource-poor mobile devices with powerful clouds : architectures, challenges, and applications. *IEEE Wireless Commun.*, *20*(3), 14–22.

Liu, X., Yu, J., Feng, Z. et Gao, Y. (2020). Multi-agent reinforcement learning for resource allocation in iot networks with edge computing. *China Commun.*, *17*(9), 220–236.

Lyu, X., Ren, C., Ni, W., Tian, H. et Liu, R. P. (2018). Distributed optimization of collaborative regions in large-scale inhomogeneous fog computing. *IEEE J. Select. Areas Commun.*, *36*(3), 574–586.

Maamar, Z., Baker, T., Faci, N., Ugljanin, E., Al-Khafajiy, M. et Burégio, V. (2019). Towards a Seamless Coordination of Cloud and Fog : Illustration

through the Internet-of-Things. Dans *Proc. 34th ACM/SIGAPP Symp. on Applied Comput.*, 2008–2015.

Mahmud, R., Kotagiri, R. et Buyya, R. (2018). Fog computing : A taxonomy, survey and future directions. In *Internet of everything* 103–130. Springer.

Manvi, S. S. et Shyam, G. K. (2014). Resource Management for Infrastructure as a Service (IaaS) in Cloud Computing : A Survey. *J. Net. and Comput. Appl.*, *41*, 424–440.

Marín-Tordera, E., Masip-Bruin, X., García-Almiñana, J., Jukan, A., Ren, G.-J. et Zhu, J. (2017). Do we all really know what a fog node is ? current trends towards an open definition. *Computer Commun.*, *109*, 117–130.

Masip-Bruin, X., Marin-Tordera, E., Jukan, A. et Ren, G.-J. (2018). Managing Resources Continuity from the Edge to the Cloud : Architecture and Performance. *Future Gen. Comput. Syst.*, *79*, 777–785.

Merkel, D. (2014). Docker : Lightweight Linux containers for consistent development and deployment. *Linux J.*, (239), 2.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I. *et al.* (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv :1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J. *et al.* (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J. et Polakos, P. A. (2018). A comprehensive survey on fog computing : State-of-the-art and research challenges. *IEEE Commun. Surv. Tuts.*, *20*(1), 416–464.

Mseddi, A., Jaafar, W., Elbiaze, H. et Ajib, W. (2019). Joint container placement and task provisioning in dynamic fog computing. *IEEE IoT J.*, *6*(6), 10028–10040.

Mseddi, A., Jaafar, W., Elbiaze, H. et Ajib, W. (2020). Intelligent resource allocation in dynamic fog computing environments. Dans *Proc. IEEE Int. Conf. Cloud Net. (CloudNet)*, 1–7.

Mukherjee, M., Shu, L. et Wang, D. (2018). Survey of fog computing : Fundamental, network applications, and research challenges. *IEEE Commun. Surv. Tuts.*, *20*(3), 1826–1857.

Naha, R. K. et Garg, S. (2021). Multi-criteria–based dynamic user

behaviour–aware resource allocation in fog computing. *ACM Trans. on Internet of Things*, *2*(1), 1–31.

Naha, R. K., Garg, S., Chan, A. et Battula, S. K. (2020). Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment. *Future Generation Computer Systems*, *104*, 131–141. Récupéré de `https://www.sciencedirect.com/science/article/pii/S0167739X19319016`

Neary, P. (2018). Automatic hyperparameter tuning in deep convolutional neural networks using asynchronous reinforcement learning. Dans *2018 IEEE Int. Conf. on Cogn. Comput. (ICCC)*, 73–77. `http://dx.doi.org/10.1109/ICCC.2018.00017`

News, I. (2017). Number of connected iot devices to hit 125 billion by 2030, says ihs markit. *Internet of Things News*. Accessed : date-of-access. Récupéré de `https://www.iottechnews.com/news/2017/oct/26/number-connected-iot-devices-125-billion-2030-ihs-markit/`

Ning, Z., Dong, P., Wang, X., Guo, L., Rodrigues, J. J. P. C., Kong, X., Huang, J. et Kwok, R. Y. K. (2019a). Deep reinforcement learning for intelligent internet of vehicles : An energy-efficient computational offloading scheme. *IEEE Trans. Cogn. Commun. Netw..*, *5*(4), 1060–1072.

Ning, Z., Huang, J. et Wang, X. (2019b). Vehicular fog computing : Enabling real-time traffic management for smart cities. *IEEE Wireless Commun.*, *26*(1), 87–93.

Ningning, S., Chao, G., Xingshuo, A. et Qiang, Z. (2016). Fog computing dynamic load balancing mechanism based on graph repartitioning. *China Commun.*, *13*(3), 156–164.

Oueida, S., Kotb, Y., Aloqaily, M., Jararweh, Y. et Baker, T. (2018). An Edge Computing Based Smart Healthcare Framework for Resource Management. *Sensors*, *18*(12), 4307.

Oueis, J., Strinati, E. C., Sardellitti, S. et Barbarossa, S. (2015). Small cell clustering for efficient distributed fog computing : A multi-user case. Dans *2015 IEEE 82nd Vehicular Technology Conf. (VTC2015-Fall)*, 1–5. IEEE.

Prateeksha, V. et Yogesh, S. (2017). Demystifying fog computing : Characterizing architectures, applications and abstractions. Dans *Proc. IEEE 1st Int. Conf. Fog and Edge Comput. (ICFEC)*, 115–124.

Pu, L., Chen, X., Xu, J. et Fu, X. (2016). D2D fogging : An energy-efficient and incentive-aware task offloading framework via network-assisted D2D

collaboration. *IEEE J. Sel. Areas Commun.*, *34*(12), 3887–3901.

Rahman, G. M. S., Dang, T. et Ahmed, M. (2020). Deep reinforcement learning based computation offloading and resource allocation for low-latency fog radio access networks. *Intelligent and Converged Netw.*, *1*(3), 243–257.

Ranjit, M. P., Ganapathy, G., Sridhar, K. et Arumugham, V. (2019). Efficient deep learning hyperparameter tuning using cloud infrastructure : Intelligent distributed hyperparameter tuning with bayesian optimization in the cloud. Dans *Proc. IEEE 12th Int. Conf. Cloud Comp. (Cloud)*, 520–522.

Rappaport, T. S. *et al.* (1996). *Wireless Communications : Principles and Practice*, volume 2. Prentice Hall PTR New Jersey.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N. et Whiteson, S. (2018). QMIX : Monotonic value function factorisation for deep multi-agent reinforcement learning. Dans *Proc. 37th Int. Conf. Mach. Learn. (ICML)*.

Rejiba, Z., Masip-Bruin, X. et Marín-Tordera, E. (2019). Computation task assignment in vehicular fog computing : A learning approach via neighbor advice. Dans *Proc. IEEE Int. Symp. Net. Comput. Appl. (NCA)*, 1–5.

Rimal, B. P., Van, D. P. et Maier, M. (2018). *Fog Computing : Principles, Architectures, and Applications*. Cham, Switzerland : Springer.

Ross, G. T. et Soland, R. M. (1975). A Branch and Bound Algorithm for the Generalized Assignment Problem. *Mathematical Programming*, *8*(1), 91–103.

Salahuddin, M. A., Al-Fuqaha, A. et Guizani, M. (2016). Reinforcement learning for resource provisioning in the vehicular cloud. *IEEE Wireless Commun.*, *23*(4), 128–135.

Santo, W. d. E., Matos Júnior, R. d. S., Ribeiro, A. d. R. L., Silva, D. S. et Santos, R. (2019). Systematic mapping on orchestration of container-based applications in fog computing. Dans *2019 15th Int. Conf. on Netw. and Service Management (CNSM)*, 1–7. `http://dx.doi.org/10.23919/CNSM46954.2019.9012677`

Satyanarayanan, M., Bahl, V., Caceres, R. et Davies, N. (2009). The Case for VM-based Cloudlets in Mobile Computing. *IEEE Pervasive Comput.*, *8*(4), 14–23.

Schwefel, H.-P. (1981). *Numerical optimization of computer models*. John Wiley & Sons, Inc.

Skarlat, O., Schulte, S., Borkowski, M. et Leitner, P. (2016). Resource Provisioning

for IoT Services in the Fog. Dans *Proc. 9th IEEE Int. Conf. on Service-Oriented Comp. and App. (SOCA)*, 32–39.

Smairi, N. (2013). *Optimisation par essaim particulaire : adaptation de tribes à l'optimisation multiobjectif.* (Thèse de doctorat). Paris Est.

Sookhak, M., Yu, F. R., He, Y., Talebian, H., Safa, N. S., Zhao, N., Khan, M. K. et Kumar, N. (2017). Fog Vehicular Computing : Augmentation of Fog Computing Using Vehicular Cloud Computing. *IEEE Veh. Tech. Mag.*, *12*(3), 55–64. http://dx.doi.org/10.1109/MVT.2017.2667499

Souza, V. B., Pereira, M. H., Lelis, L. H. S. et Masip-Bruin, X. (2020). Enhancing resource availability in vehicular fog computing through smart inter-domain handover. Dans *GLOBECOM 2020 - 2020 IEEE Global Commun. Conf.*, 1–6. http://dx.doi.org/10.1109/GLOBECOM42002.2020.9322238

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K. et Graepel, T. (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward. Dans *Proc. 17th Int. Conf. Auton. Agents and MultiAgent Syst. (AAMAS)*, p. 2085–2087., Richland, SC.

Sutton, R. S. et Barto, A. G. (1998). Reinforcement learning : An introduction. adaptive computations and machine learning. *A Bradford Book*, *1*.

Talaat, F. M. (2022). Effective prediction and resource allocation method (epram) in fog computing environment for smart healthcare system. *Multimedia Tools and Appl.*, *81*(6), 8235–8258.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J. et Vicente, R. (2017). Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, *12*(4), e0172395.

Tan, M. (1993). Multi-agent reinforcement learning : Independent vs. cooperative agents. Dans *Proc. 10th Int. Conf. Mach. Learn. (ICML)*, 330–337. Morgan Kaufmann.

Tang, Z., Zhou, X., Zhang, F., Jia, W. et Zhao, W. (2019). Migration modeling and learning algorithms for containers in fog computing. *IEEE Trans. Serv. Comput.*, *12*(5), 712–725.

Tokic, M. et Palm, G. (2011). Value-difference based Exploration : Adaptive Control between Epsilon-greedy and Softmax. Dans *Annual Conf. on Artificial Intelligence*, 335–346. Springer.

Tran-Dang, H., Bhardwaj, S., Rahim, T., Musaddiq, A. et Kim, D.-S. (2022).

Reinforcement learning based resource management for fog computing environment : Literature review, challenges, and open issues. *Journal of Commun. and Netw.*, *24*(1), 83–98. http://dx.doi.org/10.23919/JCN.2021.000041

Tran-Dang, H. et Kim, D.-S. (2021a). A distributed resource allocation algorithm for task offioading in fog-enabled iot systems. Dans *2021 Twelfth Int. Conf. on Ubiquitous and Future Networks (ICUFN)*, 455–460. http://dx.doi.org/10.1109/ICUFN49451.2021.9528792

Tran-Dang, H. et Kim, D.-S. (2021b). Task priority-based resource allocation algorithm for task offloading in fog-enabled iot systems. Dans *2021 Int. Conf. on Information Networking (ICOIN)*, 674–679. IEEE.

Vaquero, L. M. et Rodero-Merino, L. (2014). Finding your way in the fog : Towards a comprehensive definition of fog computing. *SIGCOMM Comput. Commun. Rev.*, *44*(5), 27–32.

Varghese, B., Wang, N., Nikolopoulos, D. S. et Buyya, R. (2017). Feasibility of fog computing. *arXiv preprint arXiv :1701.05451*.

Wang, K., Tan, Y., Shao, Z., Ci, S. et Yang, Y. (2019a). Learning-based task offloading for delay-sensitive applications in dynamic fog networks. *IEEE Trans. Veh. Tech.*, *68*(11), 11399–11403.

Wang, K., Yin, H., Quan, W. et Min, G. (2018a). Enabling collaborative edge computing for software defined vehicular networks. *IEEE Netw.*, *32*(5), 112–117.

Wang, L., Jiao, L., Li, J. et Mühlhauser, M. (2017). Online resource allocation for arbitrary user mobility in distributed edge clouds. Dans *Proc. IEEE 37th Int. Conf. Dist. Comput. Syst. (ICDCS)*, 1281–1290. http://dx.doi.org/10.1109/ICDCS.2017.30

Wang, N., Varghese, B., Matthaiou, M. et Nikolopoulos, D. S. (2017). ENORM : A framework for edge node resource management. *IEEE Trans. Serv. Comput.*

Wang, X., Ning, Z. et Wang, L. (2018b). Offloading in Internet of vehicles : A fog-enabled real-time traffic management system. *IEEE Trans. on Ind. Informatics*, *14*(10), 4568–4578. http://dx.doi.org/10.1109/TII.2018.2816590

Wang, Y., Wang, K., Huang, H., Miyazaki, T. et Guo, S. (2019b). Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications. *IEEE Trans. Indus. Inform.*, *15*(2), 976–986.

Waqas, M., Niu, Y., He, J., Ahmed, M., Chen, X., Li, Y., Jin, D. et Han, Z. (2019). Mobility-aware fog computing in dynamic environments : Understandings and implementation. *IEEE Access*, *7*(3), 38867–38879.

Watkins, C. J. C. H. et Dayan, P. (1992). Q-learning. Dans *Machine Learning*, 279–292.

Wen, X., Zha, Z.-J., Wang, Z., Zhuang, L. et Li, H. (2018). Ccnet : Cluster-coordinated net for learning multi-agent communication protocols with reinforcement learning. Dans *Asian Conf. on Machine Learning*, 582–597. PMLR.

Wu, Q., Liu, H., Wang, R., Fan, P., Fan, Q. et Li, Z. (2020). Delay-sensitive task offloading in the 802.11p-based vehicular fog computing systems. *IEEE IoT J.*, *7*(1), 773–785.

Xin, J., Chen, G. et Hai, Y. (2009). A Particle Swarm Optimizer with Multi-stage Linearly-decreasing Inertia Weight. Dans *Proc. IEEE Int. Joint Conf. Comput Sc. and Optimiz. (CSO)*, volume 1, 505–508. IEEE.

Xu, C., Wang, Y., Zhou, Z., Gu, B., Frascolla, V. et Mumtaz, S. (2018). A Low-Latency and Massive-Connectivity Vehicular Fog Computing Framework for 5G. Dans *Proc. IEEE Glob. Wrkshps. (Globecom Wrkshps.)*, 1–6.

Yang, M., Zhu, H., Wang, H., Koucheryavy, Y., Samouylov, K. *et al.* (2021). An online learning approach to computation offloading in dynamic fog networks. *IEEE IoT J.*, *8*(3), 1572–1584. `http://dx.doi.org/10.1109/JIOT.2020.3015522`

Yannuzzi, M., Milito, R., Serral-Gracià, R., Montero, D. et Nemirovsky, M. (2014). Key Ingredients in an IoT Recipe : Fog Computing, Cloud Computing, and More Fog Computing. Dans *Proc. IEEE 19th Int. Workshop on Comput. Aided Model. and Design of Commun. Links and Net. (CAMAD)*, 325–329.

Ye, D., Wu, M., Tang, S. et Yu, R. (2016). Scalable fog computing with service offloading in bus networks. Dans *2016 IEEE 3rd Int. Conf. on Cyber Security and Cloud comp. (CSCloud)*, 247–251. IEEE.

Yelina, T., Mylnikov, V. et Bezzateev, S. (2020). Optimal allocation of cloud service resources using multi-agent technologies. Dans *2020 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF)*, 1–4. IEEE.

Yi, S., Li, C. et Li, Q. (2015). A Survey of Fog Computing : Concepts, Applications and Issues. Dans *Proc. ACM Workshop on Mob. Big Data*, 37–42.

Yu, X.-M., Xiong, X.-Y. et Wu, Y.-W. (2004). A PSO-based Approach to Optimal Capacitor Placement with Harmonic Distortion Consideration. *Electric Power Systems Research*, *71*(1), 27–33.

Zeng, D., Gu, L., Guo, S., Cheng, Z. et Yu, S. (2016). Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Trans. on Computers*, *65*(12), 3702–3712.

Zhang, Q., Zhani, M. F., Boutaba, R. et Hellerstein, J. L. (2013). Harmony : Dynamic Heterogeneity-aware Resource Provisioning in the Cloud. Dans *Proc. IEEE 33rd Int. Conf. on Dist. Comput. Syst. (ICDCS)*, 510–519.

Zhang, Y., Yao, J. et Guan, H. (2017). Intelligent cloud resource management with deep reinforcement learning. *IEEE Cloud Comput.*, *4*(6), 60–69. `http://dx.doi.org/10.1109/MCC.2018.1081063`

Zhao, J., Kong, M., Li, Q. et Sun, X. (2020). Contract-based computing resource management via deep reinforcement learning in vehicular fog computing. *IEEE Access*, *8*, 3319–3329.

Zhu, C., Tao, J., Pastor, G., Xiao, Y., Ji, Y., Zhou, Q., Li, Y. et Yla-Jaaski, A. (2019). FOLO : Latency and quality optimized task allocation in vehicular fog computing. *IEEE IoT J.*, *6*(3), 4150–4161. `http://dx.doi.org/10.1109/JIOT.2018.2875520`

Zhu, H., Wang, H., Luo, X. et Qian, H. (2018). An online learning approach to wireless computation offloading. Dans *Proc. IEEE Glob. Conf. Sig. Info. Process. (GlobalSIP)*, 678–682.