MÉMOIRE DE FIN D'ETUDES

POUR L'OBTENTION DU

DIPLOME DE MASTER EN INFORMATIQUE DÉCISIONNELLE

Déchargement décentralisé assisté par le cache dans un environnement Edge Cloud collaboratif en utilisant l'apprentissage par renforcement

---

Réalisé par :
OUACHA Chaima

Encadré par :
Pr. SAADI Youssef

Soutenu le 22 Juin 2023, Devant le jury composé de :

Pr. FAKIR Mohamed	(FST)	- Président
Pr. ZOUGAGH Hicham	(FST)	- Rapporteur
Pr. EL MOURABIT Yousef	(FST)	- Rapporteur
Pr. SAADI Youssef	(FST)	- Directeur de thèse

Année universitaire 2022 - 2023

Dédicace

C'est avec profonde gratitude et sincères mots, que je dédie cet humble travail à :

***À ma chère maman** : pour ton soutien, tes innombrables sacrifices et ton amour. Ta présence dans ma vie m'a offert une perle si rayonnante que ce que j'imaginais. Tes encouragements sont pour moi les piliers fondateurs de ce que je suis et ce que je fais. C'est avec grand honneur que je te dédie ce travail. Merci d'être si précieuse et formidable. Merci d'être présente dans ma vie.*

***À mon cher papa**, pour tes sacrifices démesurés, déployés pour guider mes pas et tes encouragements continus qui me motivent à entreprendre ce mémoire avec sérénité et confiance en moi. Que mes chers et respectueux parents acceptent, à cette occasion, mes hommages comme gage de mon profond amour et ma reconnaissance jamais interrompue.*

***À ma chère sœur aînée Ihssane**, il est impossible de trouver des mots qui puissent véritablement exprimer la profondeur de mes sentiments envers toi. Je te souhaite sincèrement un avenir rayonnant de joie et de succès.*

***À mes chers amis**, en témoignage de l'amitié sincère qui nous a liées et de bons moments qu'on a passés ensemble.Je vous souhaite un avenir plein de succès.*

***À tous les membres de ma chère famille**, à tous ceux que j'aime, à tous ceux qui m'aiment, à tous ceux qui m'ont encouragé ou aidé au long de mes études*

Chaima.

Remerciements

En premier lieu, je remercie DIEU qui m'a donné le courage et la patience pour réussir ce travail.
En deuxième lieu, je tiens à remercier chaleureusement toute personne, ayant contribué à la réalisation de ce travail, et en particulier :
M. SAADI Youssef, Professeur chercheur à la Faculté des Science et Technique, mon cher encadrant pour ses orientations, ses corrections, ses remarques pertinentes et son soutien permanent tout au long de la période de mon PFE, merci encore une fois.
Le corps enseignant du département Informatique de la Faculté des Science et Technique de Beni Mellal pour les efforts d'amélioration de notre formation.
Je remercie enfin tous **les membres de jury** d'avoir accepté d'évaluer mon travail, en espérant qu'ils y trouveront les qualités de clarté et de motivation qu'ils attendent.
Enfin merci infiniment à tous ceux qui ont contribué de près ou de loin à l'enrichissement de ce travail et qui veille au développement de la recherche scientifique.

Résumé

De nombreuses applications émergentes telles que la réalité augmentée, la reconnaissance faciale, les voitures autonomes et la santé électronique nécessitent des calculs intensifs, et les résultats traités doivent être disponibles pour l'utilisateur en quelques millisecondes.

Cependant, un déchargement efficace des tâches nécessite un cadre de gestion des ressources. De nombreuses méthodologies de déchargement existantes ne considèrent que la latence et la consommation d'énergie dans une configuration réseau prédéfinie mise en œuvre à petite échelle. De plus, l'effet de l'emplacement de l'algorithme de déchargement n'a pas été largement étudié. Ce mémoire présente un framework de déchargement en utilisant Online *Q-learning* (QL). Le cadre proposé tient compte des contraintes strictes de latence, espace d'état élevé, mobilité des utilisateurs évoluant rapidement, ressources hétérogènes et taux d'arrivée des tâches.

La recherche proposée met également en évidence l'importance de la mise en cache et présente un nouveau concept appelé "la mise en cache des conteneurs" qui met en cache les dépendances des applications populaires. Ainsi, les décisions de déchargement sont prises en compte pour minimiser la consommation d'énergie, la latence et les coûts de mise en cache. De plus, l'importance de l'emplacement de déploiement de l'algorithme de déchargement est également examinée, et une méthode distribuée est proposée.

Des simulations approfondies ont été réalisées dans un simulateur à événements discrets implémenté en Java en utilisant des profils réalistes de tâches. Les résultats des simulations et les comparaisons avec les algorithmes de référence existants ont montré des performances remarquables en termes de consommation d'énergie, d'échecs de tâches démontrant la faisabilité de l'approche proposée.

Mots clés : Edge computing, cloud computing, online Q-learning, algorithme de référence, mise en cache.

Abstract

Many emerging technologies, including *Augmented Reality* (AR), facial recognition, autonomous vehicles, and e-health applications, require intensive computational resources, and the results of these computations must be available to users within milliseconds. To address this challenge, a combination of edge computing and cloud computing can be used to distribute the computational load across connected resources. However, efficient task offloading requires a robust resource management framework.

Unfortunately, many existing offloading methodologies only consider factors such as latency and energy consumption in a predefined network configuration at a small scale. Furthermore, the impact of the offloading algorithm's location has not been extensively studied. This thesis presents a new adaptive offloading framework that utilizes Online QL. The proposed framework takes into account strict latency constraints, a large state space, dynamic user mobility, diverse resources, and task arrival rate.

The research also emphasizes the importance of caching and introduces the concept of "container caching" to store dependencies of popular applications. By leveraging container caching, offloading decisions are made to minimize energy consumption, latency, and caching costs. Additionally, the significance of the deployment location of the offloading algorithm is thoroughly examined, and a distributed offloading method is proposed.

Extensive simulations were conducted using a discrete event simulator implemented in Java, employing realistic task profiles. The simulation results and comparisons with existing benchmarking algorithms demonstrated remarkable performance in terms of energy consumption, task failures, thereby establishing the feasibility of the proposed approach.

Keywords : Edge computing, cloud computing, online Q-learning, Benchmark algorithms, caching.

Table des figures

Liste des abréviations

Table des matières

Introduction générale

Avec le développement rapide de la technologie, il y a eu une augmentation considérable du nombre de téléphones mobiles, d'ordinateurs portables, de capteurs et d'appareils électroniques au cours de la dernière décennie. Cette augmentation des gadgets peut être liée à la prolifération des applications sophistiquées telles que la reconnaissance faciale, la e-health, les jeux en réalité augmentée/virtuelle etc. Les données utilisées et traitées par ces applications [2] sont très élevées par rapport aux applications traditionnelles telles que les médias sociaux et la navigation sur le web. Par exemple, l'automatisation industrielle [3] implique une interaction fréquente des robots avec les humains et une gamme de tâches telles que l'analyse de données , la reconnaissance de signes. Les données en temps réel disponibles doivent être collectées, analysées et traitées dans l'ordre des millisecondes dans ces scénarios. Une situation similaire peut être observée dans les villes intelligentes (smart cities)[4] pour la surveillance du trafic, la détection des piétons et la reconnaissance des panneaux de signalisation. La détection précise de tout mouvement aléatoire chez les piétons et le changement des panneaux de signalisation en conséquence sont essentiels. Tout retard ou échec dans la détection des mouvements peut entraîner de graves conséquences en termes de sécurité. Dans les jeux en réalité virtuelle(*Virtual Reality* (VR)) [5], 72 trames par seconde est le minimum requis pour le bon fonctionnement d'une application.

Les mouvements de l'utilisateur doivent être capturés et traités avec précision pour une expérience immersive. De nombreuses autres applications telles que la surveillance des patients, la reconnaissance biométrique, les traceurs d'inventaire sans fil et les maisons intelligentes nécessitent un traitement rapide des données. De plus, les exigences de toutes ces applications diffèrent en fonction de leur objectif. L'arrivée de la 5G [6] a rendu les applications de l'IoT existantes plus complexes avec des exigences strictes. Par conséquent, ces applications nécessitent le traitement d'une grande quantité de données dans un court intervalle de temps.
De plus, cela nécessite une quantité considérable d'énergie pour exécuter ces applications sur des appareils locaux. La capacité limitée de la batterie, de stockage et de vitesse de traitement disponible sur l'appareil de l'utilisateur limite l'exécution des applications localement. Ces données sont généralement envoyées à un cloud central pour être traitées afin de garantir la qualité de service à l'utilisateur. Le cloud central [7] est un centre de données équipé d'un espace de stockage important et d'une grande vitesse de traitement. Il héberge diverses machines virtuelles pour exécuter des applications en parallèle et réduire le temps d'attente des tâches. Cependant, ces centres de données sont généralement situés loin de l'utilisateur, et le délai de communication avec le cloud pour l'exécution des tâches est élevé [8]. Cela peut entraîner une violation des accords de niveau de service [9] pour les applications. De plus, toute perturbation dans une partie du réseau peut entraîner une panne globale interrompant toutes les opérations réseau en cours.

Cela a conduit à l'émergence du edge computing [10], où les serveurs sont situés plus près de l'utilisateur. Les serveurs Edge sont des mini-centres de données avec des ressources limitées disponibles au bord de l'utilisateur [11]. Étant donné que les serveurs Edge agissent comme des couches intermédiaires entre les appareils des utilisateurs et le cloud, les demandes de tâches sélectionnées dirigées vers le cloud peuvent être traitées par le serveur Edge. Cela réduit la charge sur le cloud central et allège la charge sur les ressources réseau pour envoyer les informations de tâche. De plus, ces serveurs répartis géographiquement sont plus adaptés pour surveiller les données en temps réel [12] et faciliter les contraintes de traitement. Par conséquent, la combinaison d'éxecution en Edge et en cloud peut servir plusieurs utilisateurs de manière efficace et garantir des réponses rapides et interactives.

Problématique

Les données massives générées par diverses applications IoT telles que les voitures connectées, les usines intelligentes et les objets connectés doivent être traitées et les résultats des utilisateurs doivent être déployés. Parfois, le traitement peut être effectué localement, mais en raison de la capacité limitée des dispositifs locaux en termes de puissance de batterie et de capacité de calcul, ces données doivent être transmises au serveur edge ou au cloud distant pour un traitement ultérieur. Cependant, les ressources disponibles sur le serveur edge sont également limitées par rapport au cloud central en raison de considérations de coût et de viabilité. De plus, certaines applications complexes nécessitent une grande quantité d'espace de stockage et de puissance de calcul qui peuvent ne pas être disponibles sur les serveurs edge, et ces tâches doivent donc être redirigées vers le cloud. L'envoi de plusieurs tâches pour un calcul cloud entraîne une augmentation des délais et une utilisation élevée de la bande passante pour la transmission des données. Par conséquent, la méthode de déchargement vers le cloud edge est un défi en raison de plusieurs facteurs, tels que la mobilité fréquente des utilisateurs, l'hétérogénéité des ressources disponibles et la fluctuation des performances du réseau. Il est donc essentiel d'identifier un emplacement approprié pour le déchargement en fonction des dynamiques et des exigences du réseau de la tâche.

Contribution

Dans ce mémoire, un algorithme d'apprentissage par renforcement est proposé pour choisir les emplacements de déchargement de nombreuses tâches générées par divers utilisateurs. L'algorithme proposé prend en compte les dépendances des tâches telles que la limite de délai autorisée, la durée de la tâche, la taille de la demande, ainsi que les ressources du réseau telles que la bande passante disponible et la puissance de calcul pour l'exécution d'une tâche. De plus, la capacité à apprendre de manière autonome les changements dynamiques dans un réseau permet à l'algorithme de surmonter les contraintes strictes de latence, la mobilité des utilisateurs changeante et les fluctuations de la bande passante disponible. De plus, l'utilisation d'une mise en cache décentralisée basée sur des conteneurs est suggérée pour exploiter les avantages de la mise en cache afin de réduire les exigences strictes de latence des futures applications IoT. Cette approche recommandée a permis

de réduire de manière significative les délais encourus et consommation d'énergie.

Organisation du mémoire

Ce mémoire est organisé comme suit :

— Le chapitre 1 fournit des informations de base sur le cloud computing, le edge computing, les scénarios de déchargement. Il explique également comment la recherche proposée surmonte les problèmes existants d'allocation des ressources et met en évidence l'importance de l'approche recommandée. Il présente ensuite les concepts du Machine Learning et du Reinforcement Learning, et aborde la façon dont le Q Reinforcement Learning peut résoudre les problèmes existants.

— Le chapitre 2 présente le modèle du système étudié et présente en évidence la formulation des contraintes, des paramètres et de la fonction objective qui vont servir pour la modélisation de notre approche basée sur le Q learning

— Le chapitre 3 propose une analyse théorique de l'utilisation du Reinforcement Learning pour ce problème. De plus, il décrit l'algorithme proposé pour obtenir les décisions de déchargement pour diverses tâches en fonction de la latence, de la consommation d'énergie et du coût de mise en cache.

— Le chapitre 4 présente les scénarios de simulation de notre approche ainsi que les résultats obtenus.

Chapitre 1

Contexte général et état de l'art

1.1 Introduction

Ce chapitre décrit l'état de l'art des techniques du déchargement et de gestion des ressources en edge computing. La section 1.2 fournit des informations de base sur le cloud computing, le edge computing, les avantages de l'environnement collaboratif du edge cloud et le scénario du déchargement des tâches. La section 1.2.5 introduit les concepts de base de l'apprentissage par renforcement ainsi que ses types et applications. La section 1.2.6 presente des informations détaillées sur le *Q-Reinforcement learning* (QRL) qui sera utilisé dans cette recherche.

1.2 Contexte général

L'énorme croissance [13] du nombre d'ordinateurs portables, de mobiles, de portables intelligents, de capteurs d'internet des objets, peut être liée aux améliorations significatives dans l'industrie des communications, telles que l'augmentation des taux de données et la diversification des produits pour différentes applications. Avec le développement rapide de gadgets électroniques, diverses applications telles que les jeux en ligne, e-santé, vidéosurveillance, voitures connectées ont émergé. La plupart de ces applications fonctionnent de manière intégrée pour atteindre la fonctionnalité souhaitée.

L'idée de cette technologie connectée est d'être populaire jour après jour, et les principales raisons incluent la détection, l'intégration, l'analyse et le contrôle en temps réel des données. Par exemple, les capteurs et les caméras travaillent de manière collaborative pour détecter et identifier tout obstacle dans la surveillance intelligente de la circulation. Les données agrégées sont traitées pour faire des inférences sur la distance et l'emplacement de l'obstacle. Ces observations servent ensuite à contrôler les feux de circulation. Le concept de villes intelligentes peut également être compris en corrélant avec l'exemple précédent, mais la quantité de données traitées et analysées est beaucoup plus élevée. Par conséquent, l'idée principale dans les services intelligents est de réaliser

la fonctionnalité souhaitée pour n'importe quelle application avec le contrôle le plus négligeable, et le nombre augmente rapidement.

En outre, le développement de services connectés [14] tels que les villes intelligentes, etc augmentera inévitablement le nombre d'applications IoT utilisées au jour le jour. Une telle hausse rapide des applications augmente la quantité de données générées et les calculs nécessaires pour gérer les applications.

Etant donné la petite taille, la faible consommation d'énergie et la capacité de traitement limitée d'un appareil utilisateur, les applications à forte intensité de traitement ne peuvent pas être exécuté localement. Leur traitement local peut enfreindre les délais et épuiser la batterie rapidement.Par conséquent, les données générées par différentes applications sont généralement transmises aux cloud central [15] pour traitement.

1.2.1 Cloud computing

Les serveurs de cloud centralisés prennent en charge une variété de services [16] tels que *Infrastructure as a Service* (IaaS), *Software as a Service* (SaaS), Database as a Service (DaaS), Information as a Service, and Platform as a Service, cela dépend des besoins des différentes applications. Ces puissants centres de données, dotés de capacités de traitement importantes, ont été bénéfiques pour de nombreuses applications au fil des années. Ces centres de données cloud hébergent de nombreuses machines virtuelles avec un espace de stockage suffisant et une capacité de calcul élevée pour exécuter des tâches en parallèle. Ainsi, le traitement des applications dans le cloud a contribué à réduire considérablement la consommation d'énergie des appareils utilisateurs. De plus, de nombreux capteurs IoT ne disposent pas de la possibilité de traiter les données localement.

Par conséquent, les données enregistrées par les capteurs sont généralement envoyées vers le cloud distant pour une analyse ultérieure, et des mesures précises sont prises en fonction des résultats prévus. Néanmoins, le problème majeur est que ces clouds distants sont généralement situés loin de l'utilisateur. Par conséquent, cela entraîne un délai de service considérable composé de retards de transport et de transmission [17] pour transférer les données d'application vers le cloud. Parfois, le délai de service peut dépasser le délai autorisé associé à une tâche spécifique, ce qui déprécie la qualité de la livraison pour diverses applications [18]. Par exemple, une application de vision par ordinateur qui détecte les objets dans une application de voiture autonome nécessite une latence de quelques millisecondes pour garantir la sécurité [19]. Cela rend le cloud souvent inadapté à l'exécution de tâches

Cependant, les applications émergentes de la 5G avec une communication ultra-fiable à faible latence, une large bande passante mobile améliorée et une communication de type machine massive imposent des contraintes de latence strictes avec une grande vitesse de traitement. Ces applications sensibles à la latence et intensives en calcul posent un défi majeur pour respecter les accords de

niveau de service et sont résilientes à la scalabilité lorsqu'on utilise des clouds distants pour traiter les données. Par conséquent, le cloud computing traditionnel seul ne parvient pas à répondre aux exigences rigoureuses en termes de latence imposées par les applications 5G [20] . De plus, un nombre accru d'utilisateurs essayant d'accéder au cloud pour traiter leurs données augmente la charge sur la bande passante disponible et affecte les performances du réseau, comme le montre la figure 1.1. Toute perturbation du réseau affecte l'exécution globale des tâches et peut entraîner une panne parfois.

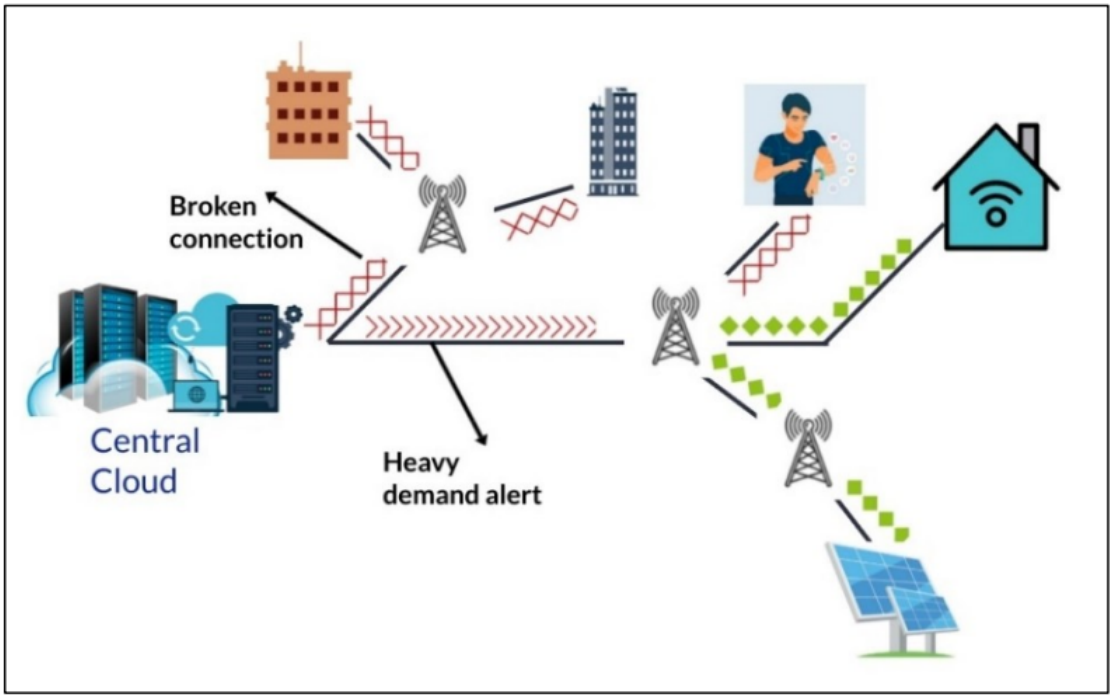

FIGURE 1.1 – Scénario d'exécution de tâches de cloud centralisé [1]

1.2.2 Edge Computing

Le concept du edge computing a été récemment proposé pour surmonter les défis mentionnés précédemment. Dans le edge computing, les edge servers sont situés plus près de l'utilisateur pour améliorer leur expérience en incorporant des ressources de calcul proches à lui. Le edge server est un mini-centre de données avec une capacité de stockage et de traitement limitée pour l'exécution des tâches. Ces serveurs sont généralement répartis géographiquement pour réduire la charge sur le cloud distant, permettre l'équilibrage de charge et assurer un réseau robuste [21] . La capacité à exécuter des services au bord du réseau en appliquant les concepts du cloud computing est appelée Mobile-Edge-Computing (MEC) par l'Institut européen des normes de télécommunications (ETSI) en 2014 [22] , et les principales caractéristiques sont les suivantes :

a) **Sur site** : Les serveurs Edge peuvent fonctionner dans des environnements isolés en utilisant les ressources disponibles localement. Ce scénario est utile pour des systèmes tels que le

secteur bancaire où la sécurité et la confidentialité sont essentielles.

b) **Proximité** : Comme les serveurs Edge sont sont situés plus près de l'utilisateur,il est facile de surveiller, d'analyser et d'obtenir des informations en temps réel. De plus, les informations mises à jour peuvent être directement utilisées pour effectuer des actions souhaitées pour diverses applications en ligne.

c) **Connaissance de l'emplacement** : La présence voisine de serveurs edge permet aux utilisateurs de profiter de services basés sur l'emplacement tels que les restaurants et les hôpitaux à proximité. De plus, les utilisateurs ont accès à des informations et divertissements locaux en fonction de leur emplacement.

d) **Faible latence** : Le délai de communication nécessaire pour transmettre les données requises et exécuter les applications peut être réduit en les traitant sur les serveurs edge. De plus, l'utilisation de la bande passante est également réduite grâce à la proximité des serveurs edge. En conséquence, la qualité de livraison et la congestion du réseau peuvent être améliorées en incorporant ces serveurs.

e) **Information sur le contexte du réseau** : L'avantage de la proximité des serveurs edge peut être mis à profit en agrégeant et en utilisant les données du réseau ainsi que les informations contextuelles locales. La qualité de l'expérience pour divers utilisateurs peut être monétisée en utilisant les inférences des données enregistrées et en fournissant des services personnalisés.

Un serveur edge peut être une unité routière qui récupère des données à partir de voitures à proximité, ou il peut être une passerelle dans une maison intelligente, etc. Ces serveurs edge agissent comme une couche intermédiaire entre les appareils utilisateurs et le cloud en calculant les résultats en traitant et analysant les données générées. De cette manière, le trafic réseau peut être réduit en réduisant la quantité de données transférées vers le cloud. Par exemple, plusieurs capteurs sont installés dans une grille solaire pour surveiller les performances du système. Ces capteurs génèrent chaque jour une quantité massive de données, dont la plupart concernent leur fonctionnement stable. Envoyer l'ensemble de ces données fréquemment vers le cloud distant n'est pas très utile et augmente l'utilisation de la bande passante. Au lieu de cela, le serveur edge local peut traiter les données générées, et un rapport résumé ou seulement les défauts peuvent être communiqués au cloud distant pour une analyse ultérieure. De cette manière, les serveurs edge géographiquement répartis réduisent la traversée de données et allègent la charge sur le cloud distant, comme le montre la figure 1.2.

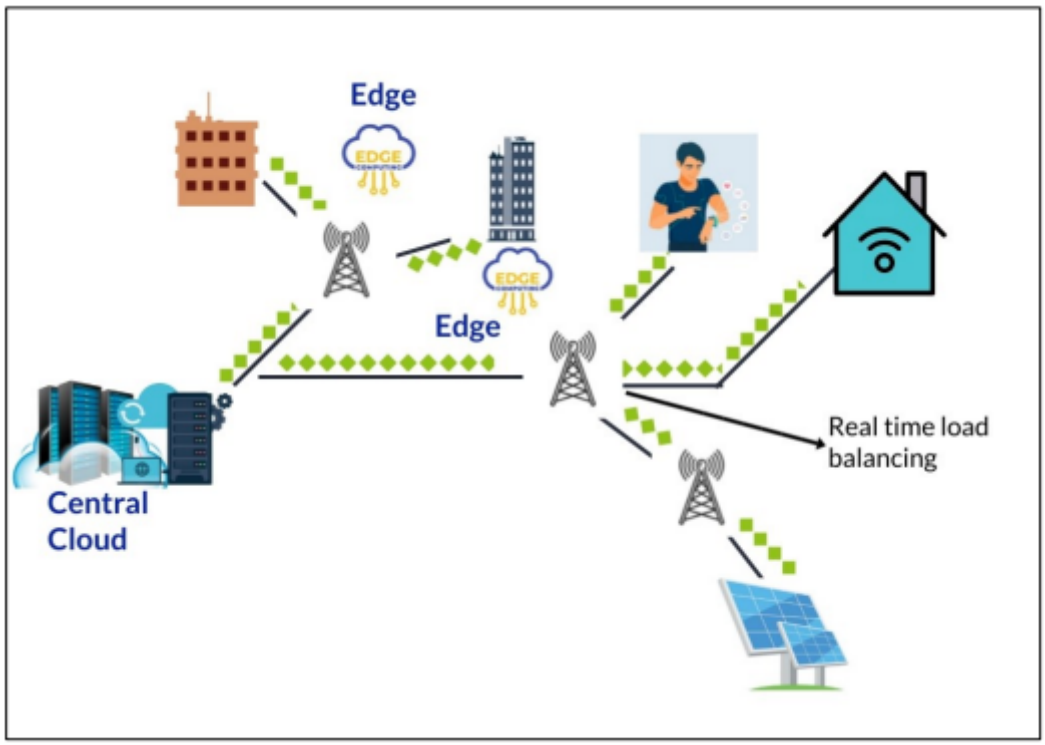

FIGURE 1.2 – L'exécution de tâches en utilisant les serveurs cloud et edge [1]

Les principaux avantages d'incorporer des serveurs edge en complément du cloud distant peuvent être résumés comme suit :

a) **La latence** : Le temps total pour traiter les données et renvoyer les résultats à l'utilisateur est connu sous le nom de latence. Si le traitement ne se produit pas sur l'appareil utilisateur, alors le temps de transfert des informations d'entrée et de sortie vers et depuis le centre de données doit également être ajouté à la latence. Certaines applications peuvent avoir des accords de niveau de service concernant la latence acceptable. Par conséquent, la latence est un facteur crucial pour l'exécution des tâches. Étant donné que le serveur edge est plus proche de l'utilisateur que le cloud distant, le retard dans le traitement des applications est faible.

b) **La bande passante disponible** : La bande passante disponible détermine la vitesse de transmission de données sans fil entre deux appareils. Par conséquent, la bande passante doit être utilisée efficacement pour obtenir des performances réseau décentes sans interruptions. Dans un modèle de cloud centralisé autonome, les transferts de données fréquents augmentent la charge sur la bande passante disponible. En revanche, dans un modèle de traitement edge, des décisions intelligentes peuvent être prises en analysant les données générées localement et en ne transférant que les informations nécessaires vers le cloud distant. Par conséquent, l'utilisation de serveurs edge rend le réseau robuste et permet l'équilibrage de charge.

c) **La scalabilité** : L'augmentation du nombre de dispositifs IoT peut conduire à la prolifération

d'applications connectées. De plus, une forte augmentation du nombre d'appareils utilisateurs interconnectés produit d'énormes quantités de données qui nécessitent un traitement en temps réel. La bande passante limitée disponible et le temps d'attente élevé dans le cloud peuvent restreindre le nombre d'applications pouvant être traitées instantanément. Par conséquent, des problèmes de scalabilité de charge, de gestion de réseau et d'exécution de tâches se posent lorsqu'un seul serveur cloud centralisé administre l'ensemble du réseau.

d) **La sécurité et la confidentialité des données** : Le transfert d'informations sur une longue distance peut augmenter l'exposition des données. Cela peut conduire à des violations de sécurité ou à une mauvaise manipulation par des pirates informatiques. Par conséquent, la transmission de données sur de courtes distances via des serveurs sécurisés améliore la confidentialité du réseau.

Cependant, les ressources limitées disponibles au niveau de l'edge peuvent limiter le nombre de tâches traitées dans un temps donné, ou le serveur edge peut ne pas posséder une capacité de stockage ou de traitement suffisante. Cette situation nécessite la diversion de certaines tâches vers le cloud pour un traitement plus rapide. De plus, certaines tâches générées par des applications sont assez petites et peuvent être traitées sur un appareil local en fonction de la disponibilité des ressources. Par conséquent, pour utiliser tous les avantages mentionnés précédemment, il est nécessaire d'avoir un environnement edge cloud collaboratif pour une gestion efficace des ressources. Ainsi, il est essentiel de déterminer l'emplacement de l'exécution des tâches pour diverses tâches pour le bon fonctionnement d'un réseau.

1.2.3 Scénarios de déchargement des tâches

Toute tâche générée peut être exécutée localement, sur un serveur edge ou sur un cloud central, comme le montre la figure 1.3. Le processus décisionnel d'un lieu d'exécution pour diverses tâches est difficile et d'une importance primordiale pour diverses raisons :

Premierement, les centres de données disponibles en edge du réseau, en cloud distant et en serveur local diffèrent considérablement en espace de stockage, vitesse de traitement, machines virtuelles disponibles pour une exécution parallèle. De plus, il convient de veiller à ce que la batterie de l'appareil de l'utilisateur ne soit pas trop sollicitée pour exécuter des applications localement. Une utilisation excessive des appareils locaux pour l'exécution peut vider rapidement la batterie et entraîner une dépréciation de la qualité de service.

Deuxièmement, la consommation d'énergie varie selon la durée de la tâche et le lieu de déchargement. La consommation d'énergie pour l'exécution d'une tâche comprend trois parties : envoyer des informations liées à la tâche au serveur choisi ; exécuter les applications ; renvoyer les résultats à l'utilisateur après exécution. Troisièmement, le délai de transfert de l'information liée aux tâches varie selon l'emplacement des centres de données et des appareils utilisateurs disponibles.

Quatrièmement, la bande passante du réseau disponible varie à tout moment, provoquant des interférences pour la transmission des données. En plus de tous ces facteurs, de nombreux utilisateurs génèrent des tâches simultanément à tout moment et certains utilisateurs mobiles peuvent être confrontés à de fréquentes interruptions du réseau.

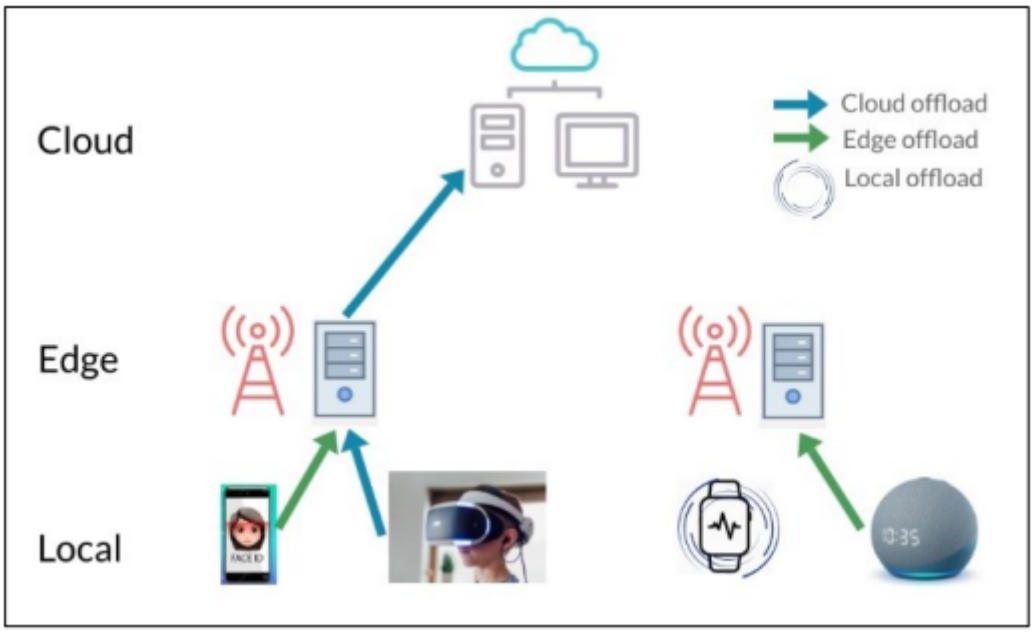

FIGURE 1.3 – Scénario de déchargement des tâches [1]

Par conséquent, le scénario dans lequel plusieurs appareils d'utilisateurs génèrent des tâches complexes, modifient instantanément les paramètres du réseau et les ressources fluctuantes rendent ce système à espace d'état élevé plus complexe.

En plus des contraintes liées aux ressources évoquées ci-dessus, les paramètres d'une tâche jouent également un rôle crucial dans le choix de l'emplacement d'exécution. Par exemple, une tâche générée pour une application d'automatisation en usine exige que les résultats soient renvoyés en millisecondes, mais la durée de la tâche peut être brève. Dans les applications de smart health comme la chirurgie à distance, une bande passante élevée, des débits de données élevés et une latence ultrafaible sont obligatoires.

Tout compromis peut entraîner de graves conséquences. En revanche, les applications de jeux de réalité virtuelle peuvent générer des tâches longues avec des exigences de délai strictes. De plus, une application d'infodivertissement peut avoir des exigences de latence détendues mais demander une sortie de grande taille. Pour cette raison, les exigences et les spécifications relatives aux tâches doivent également être prises en considération lors de la détermination de l'emplacement d'une tâche dans n'importe quel scénario.

Il est donc essentiel de décider efficacement le lieu d'exécution des tâches pour le bon fonctionnement du réseau. Le processus de déchargement pour sélectionner un emplacement optimal pour l'exécution des tâches peut se résumer comme suit :

a) Les paramètres liés à la tâche tels que la longueur (en millions d'instructions), le délai autorisé, la taille d'entrée, les dépendances logicielles doivent être collectés. En plus des informations liées à la tâche, les ressources disponibles au centre de données doivent également être rassemblées. Cela inclut la capacité de traitement locale, le serveur edge, la capacité de traitement en cloud distant, le nombre de machines virtuelles inactives, les cœurs disponibles, le délai pour y accéder. Enfin, des informations liées au réseau doivent également être collectées pour les transférer à l'emplacement concerné.

b) Le type de tâche, comme l'intensité de traitement ou la sensibilité aux délais, doit être déterminé selon les paramètres associés à la tâche. Il convient également de vérifier que l'exécution locale est suffisante pour les tâches à court terme. En outre, certains appareils locaux, comme les capteurs, n'ont pas la capacité de fonctionner localement, et la situation exige une réorientation des tâches. Par conséquent, il faut filtrer les sites de traitement potentiels en fonction de la classification des tâches et de l'analyse de faisabilité.

c) Enfin, l'emplacement d'exécution de la tâche est décidé sur la base d'un cadre d'optimisation contraint qui atteint les objectifs souhaités tels que la latence minimale, une consommation d'énergie réduite ou une bande passante réseau équilibrée. De plus, un créneau horaire pour planifier la tâche est choisi, ainsi qu'un mode de transmission.

Par conséquent, il est crucial de décider l'emplacement d'exécution de toute application en fonction de la disponibilité des ressources, des contraintes de latence, de la puissance de la batterie et des exigences de la tâche. Au fur et à mesure que nous avancerons, le nombre d'appareils connectés augmentera de façon exponentielle [13] et les tâches générées par ces appareils augmenteront également, ce qui accroîtra la complexité. De plus, afin d'atteindre une méthodologie de déchargement optimale, nous utiliserons le Machine learning , plus précisement le Reinforcement. Cette approche nous permettra d'obtenir une exécution plus rapide, plus économique et plus stable des tâches. En plus de considérer l'énergie consommée et le délai encouru pour traiter une application, le stockage des données fréquemment utilisées (la mise en cache) peut également améliorer les performances du réseau en évitant le téléchargement répété d'informations, comme expliqué ci-dessous :

1.2.4 La mise en cache

Selon les prévisions de Cisco [23] , le trafic de données mobile devrait augmenter jusqu'à 800% au cours des cinq prochaines années. L'utilisation intensive des applications IoT et leurs exigences strictes imposent une charge importante sur la bande passante disponible et affectent la qualité de service. Par conséquent, l'utilisation de la mémoire cache d'un serveur edge est recommandée pour améliorer le réseau. Les données fréquemment utilisées peuvent être stockées dans le stockage interne ou la RAM en fonction des exigences de temps d'accès, connues sous le nom de mise en cache.

Les dépendances logicielles et les bibliothèques requises sont téléchargées depuis un serveur centralisé pour traiter avec succès toute tâche d'un type d'application sur un serveur edge. Lorsque plusieurs utilisateurs demandent des tâches du même type d'application, ces fichiers de configuration doivent être téléchargés à plusieurs reprises pour exécuter l'application. Le téléchargement fréquent de contenu augmente simultanément l'utilisation de la bande passante disponible, ce qui provoque une congestion du réseau [24] . Par conséquent, le contenu populaire peut être mis en cache sur le serveur edge pour surmonter le téléchargement en double et améliorer les performances du réseau. De plus, à mesure que le coût de location d'espace de stockage diminue de jour en jour, l'incorporation d'un stockage de mise en cache sur le serveur edge est une solution efficace.

Les avantages du déploiement de serveurs edge basés sur le cache peuvent être résumés comme suit :

a) La mise en cache de contenu important sur les serveurs edge empêche les téléchargements fréquents depuis le cloud, ce qui permet de conserver une grande bande passante pour transférer les informations liées aux tâches. De plus, la popularité d'une application varie en fonction de l'emplacement. En conséquence, différents serveurs edge peuvent avoir des contenus en cache différents, ce qui améliore l'utilisation de la bande passante.

b) Stocker le contenu plus près de l'appareil de l'utilisateur empêche parfois l'envoi de la tâche vers le cloud distant pour son exécution. De cette manière, effectuer des tâches plus proches de l'utilisateur réduit la latence de service encourue lors de la livraison des résultats à l'utilisateur.

c) Le téléchargement de contenu depuis d'autres data centers ou l'envoi de tâches vers un serveur distant centralisé coûtent souvent beaucoup d'énergie. Comme il est essentiel de minimiser autant que possible la consommation d'énergie dans un problème de déchargement, le cache de contenu peut y être la solution adéquate.

d) Par ailleurs, le contenu mis en cache sur une machine virtuelle sur un serveur embarqué peut être distribué à plusieurs autres machines si nécessaire. On peut donc améliorer l'efficience spectrale en partageant le même spectre.

Par ailleurs, il est avantageux que les bibliothèques dépendantes et les fichiers de configuration requis pour exécuter une application soient mis en cache. Toutes les bibliothèques, fichiers d'initialisation, fichiers de configuration et dépendances nécessaires à l'exécution d'une application sont nommés [25] conteneurs. La conteneurisation facilite le traitement des demandes sur toute machine virtuelle. Par exemple, une application depuis une machine Linux peut fonctionner rapidement sur une machine Windows avec des conteneurs. Il est donc plus faisable de mettre en cache les fichiers de configuration des différentes applications. Les serveurs edge ont une capacité de stockage limitée allouée pour la mise en cache de données, et les dépendances logicielles des applications fréquemment demandées peuvent être mises en cache.

1.2.5 Machine Learning

Avec les améliorations des capacités des appareils électroniques en termes de matériel et de vitesse de traitement, le concept d'apprentissage automatique suscite un intérêt croissant pour de nombreuses applications telles que la détection d'objets, le défloutage d'images et l'imagerie médicale. L'apprentissage automatique est une branche de l'intelligence artificielle (IA) et de l'informatique qui se concentre sur l'utilisation de données et d'algorithmes pour imiter la façon dont les humains apprennent, en améliorant progressivement leur précision. L'apprentissage automatique est un élément important du domaine croissant de la science des données. Grâce à l'utilisation de méthodes statistiques, les algorithmes sont entraînés à effectuer des classifications ou des prédictions, et à découvrir des informations clés dans le cadre de projets d'exploration de données.[26] Ces informations orientent ensuite la prise de décision dans les applications et les entreprises, en ayant idéalement un impact sur les principaux indicateurs de croissance. À mesure que les données volumineuses continuent de se développer, la demande sur le marché pour des scientifiques des données va augmenter. Ils seront chargés d'aider à identifier les questions commerciales les plus pertinentes et les données nécessaires pour y répondre [27]. La figure 1.4 illuste la relation entre l'IA, ML et Deep Learning.

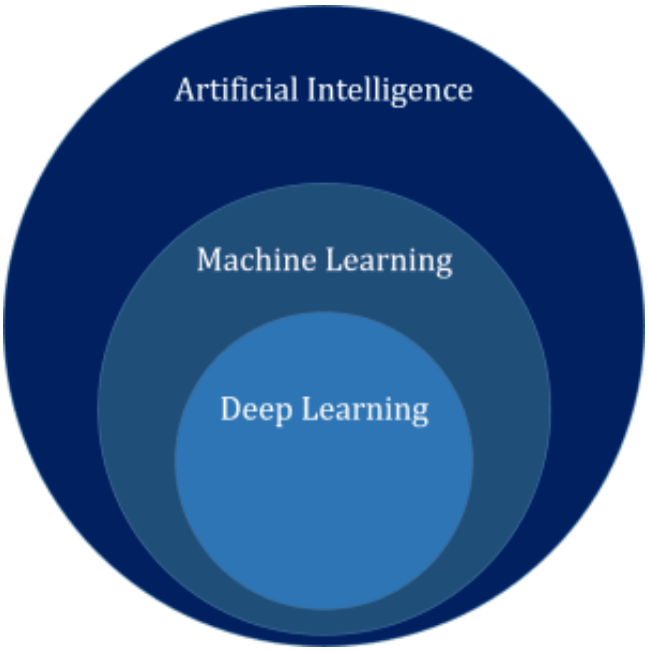

FIGURE 1.4 – Relation entre l'IA, ML et Deep Learning.[1]

a) **L'apprentissage supervisé** : Comme son nom l'indique, l'apprentissage automatique supervisé est basé sur la supervision. Cela signifie que dans la technique d'apprentissage supervisé, nous entraînons les machines à l'aide d'un ensemble de données "étiquetées", et en fonction de l'entraînement, la machine prédit la sortie. Ici, les données étiquetées indiquent que certaines des entrées sont déjà associées à la sortie. Plus précisément, nous pouvons dire que

nous entraînons d'abord la machine avec l'entrée et la sortie correspondante, puis nous demandons à la machine de prédire la sortie à l'aide de l'ensemble de données de test.

b) **L'apprentissage non supervisé** : L'apprentissage automatique non supervisé est différent de la technique d'apprentissage supervisé. Cela signifie que, dans l'apprentissage automatique non supervisé, la machine est entraînée à l'aide d'un ensemble de données non étiquetées, et la machine prédit la sortie sans aucune supervision. Dans l'apprentissage non supervisé, les modèles sont entraînés avec des données qui ne sont ni classifiées ni étiquetées, et le modèle agit sur ces données sans aucune supervision. L'objectif principal de l'algorithme d'apprentissage non supervisé est de regrouper ou catégoriser l'ensemble de données non triées en fonction des similarités, des motifs et des différences. Les machines sont instruites pour trouver les motifs cachés dans l'ensemble de données d'entrée.

c) **L'apprentissage semi-supervisé** : l'apprentissage semi-supervisé se situe entre l'apprentissage supervisé et non supervisé et fonctionne sur des données comportant quelques étiquettes, il est principalement composé de données non étiquetées. Étant donné que les étiquettes sont coûteuses, mais qu'elles peuvent être nécessaires à des fins professionnelles, elles peuvent être présentes en nombre limité. L'apprentissage semi-supervisé est complètement différent de l'apprentissage supervisé et de l'apprentissage non supervisé, car il est basé sur la présence et l'absence d'étiquettes.

d) **L'apprentissage par renforcement** : L'apprentissage par renforcement fonctionne selon un processus basé sur les retours d'information, dans lequel un agent d'intelligence artificielle (un composant logiciel) explore automatiquement son environnement en expérimentant, en prenant des actions, en apprenant de ses expériences et en améliorant ses performances. L'agent est récompensé pour chaque bonne action et puni pour chaque mauvaise action ; l'objectif de l'agent d'apprentissage par renforcement est donc de maximiser les récompenses.

   Dans l'apprentissage par renforcement, il n'y a pas de données étiquetées comme dans l'apprentissage supervisé, et les agents apprennent uniquement à partir de leurs expériences. La prochaine section va detailler le Reinforcement Learning.

1.2.6 L'apprentissage par renforcement

L'apprentissage par renforcement est un processus d'apprentissage où un modèle interagit avec son environnement. Un système de récompenses est utilisé pour renforcer les choix positifs du modèle et pénaliser les choix négatifs. L'aperçu du RL peut être vu dans la figure 1.5. Les retours collectifs sont utilisés pour entraîner ou corriger le modèle afin de prédire des sorties appropriées à l'avenir. Les sorties désirées sont obtenues par exploration dans les premières étapes et par exploitation de l'environnement connu dans les itérations suivantes. L'objectif principal dans les problèmes de RL est de choisir la meilleure action de manière à maximiser la récompense agrégée. Le mé-

canisme du reinforcement learning pour prendre la bonne décision en fonction des récompenses et des pénalités rend cet algorithme bien adapté aux environnements dynamiques. Le Processus de Décision Markovien, comme expliqué ci-dessous, est généralement utilisé pour la modélisation du RL.

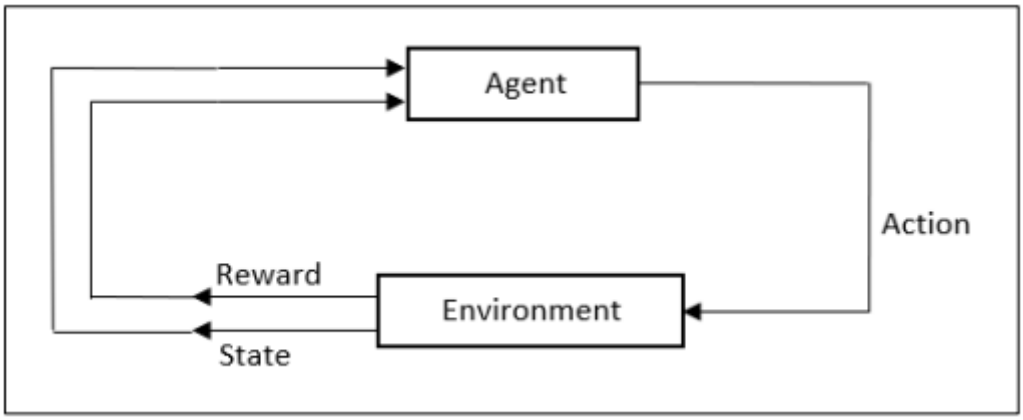

FIGURE 1.5 – Schéma du Reinfocement Learning [1]

1.2.6.1 Processus de décision markovien

Le processus de décision de Markov [28] est un cadre mathématique utilisé pour modéliser les prises de décision dans des environnements stochastiques. Les MDPs sont utilisés dans de nombreux domaines tels que la théorie des jeux, [29] l'optimisation de la gestion de projet et la robotique. Les principaux composants d'un *Markov decision process* (MDP) sont les suivants :

a) **Environement** :L'environnement dans un problème de RL (Reinforcement Learning) comprend des récompenses, des pénalités, des actions et des états possibles. L'agent interagit pour effectuer une action, et les résultats correspondants sont obtenus dans cet environnement. Cela peut être une limite à l'intérieur de laquelle un robot se déplace, un échiquier où les pièces d'échecs se déplacent, ou encore un jeu Atari Wall breaker.

b) **États** : un état représente une situation dans laquelle le système se trouve à un moment donné. Les états peuvent être déterministes ou stochastiques, c'est-à-dire que l'état suivant peut être prédit avec certitude ou être incertain.

c) **Actions** : une action est une décision prise par l'agent dans un état donné. Les actions possibles dépendent de l'état dans lequel l'agent se trouve.

d) **Fonction de transition** : la fonction de transition définit la probabilité de passer d'un état à un autre en prenant une action donnée. Cette fonction décrit la dynamique du système et est souvent représentée par une matrice de transition.

e) **Récompenses** : une récompense est un signal de feedback qui permet à l'agent d'évaluer la qualité de ses actions. La récompense peut être positive, négative ou nulle et est associée à chaque transition d'état.

f) **Politique** : la politique est la stratégie globale de l'agent pour choisir des actions en fonction de l'état actuel. Elle peut être déterministe ou stochastique et peut être représentée par une fonction qui attribue une probabilité à chaque action possible dans chaque état[28].

Avec les composants principaux ci-dessus, RL peut être appliqué à diverses applications pour améliorer les performances en fonction des expériences précédentes. Une politique optimale peut être identifiée pour passer entre différents états en effectuant des actions de sorte que la récompense soit maximisée. RL peut être catégorisé de plusieurs façons. Tout algorithme RL peut être basé sur la prédiction et le contrôle, ou il peut être basé sur la fonction de politique et de valeur.En outre, les algorithmes RL peuvent être classifiés en deux catégories : basé sur modèle et non basé sur modèle. En outre, une description détaillée de la classification du modèles basés sur RL sera élaborée dans les pages suivantes.

RL basé sur modèle : Dans les algorithmes basés sur des modèles, l'agent développe le modèle du l'environnement sur une période basée sur la transition entre les états en utilisant les actions et les récompenses. Le modèle développé est utilisé et mis à jour progressivement pendant la formation pour atteindre un objectif ou un état final. Comme toute l'information des transitions précédentes est stockée et mise à jour, la complexité des calculs est importante dans les RL basées sur les modèles, et une forte convergence n'est pas garantie. Ils sont bien adaptés aux approches axées sur les cibles, et ils apprennent par la planification [30] .

RL sans modèle : Les algorithmes sans modèle fonctionnent généralement par essais et erreurs, et ils apprennent à prendre la meilleure action à partir de l'état actuel basé sur des expériences passées. Le modèle apprend les poids et les paramètres idéaux de façon itérative, et la convergence est beaucoup plus lente par rapport au RL basé sur le modèle. Une forte convergence peut être garantie une fois que les paramètres sont appris. Les méthodes Monte-Carlo et Temporal Differences sont couramment utilisées dans les algorithmes sans modèle. Le fonctionnement de Monte Carlo est similaire à l'échantillonnage à partir d'une distribution de probabilité. L'agent apprend à atteindre l'objectif final en utilisant une politique optimale ou par la valeur en plusieurs épisodes. Cependant, le seul inconvénient de Monte Carlo est que l'agent doit attendre la fin de l'épisode pour obtenir la fonction value qui peut être utilisée plus loin. Contrairement à Monte Carlo, qui met à jour après chaque épisode, la méthode des différences temporelles fonctionne après chaque pas de temps. Cette méthode est idéale pour les environnements continus et présente une faible variance avec un certain biais. Certains des algorithmes populaires sans modèle sont SARSA et Q-learning comme indiqué ci-dessous.

Q-learning :C'est un algorithme d'apprentissage par renforcement qui permet à un agent d'apprendre à prendre des décisions en interagissant avec un environnement. L'objectif de l'agent est de maximiser une récompense cumulative au fil du temps en choisissant des actions appropriées en fonction de l'état actuel de l'environnement.[28] Le Q dans Q-learning signifie "qualité" et représente la valeur de la qualité de l'action dans un état particulier. Le Q-learning est un algorithme

hors politique, ce qui signifie que l'agent n'a pas besoin d'avoir une politique en place pour choisir ses actions. Au lieu de cela, il utilise une table Q pour stocker la qualité des actions pour chaque état possible de l'environnement.[31]

La Q-value est une récompense cumulative donnée à partir de l'état actuel (s_t) et l'action (a_t).On la note comme suit :

$$Q^{\pi}(s_t, a_t) = E(R_{t+1}|[s_t, a_t]) \tag{1.1}$$

dont E(R_{t+1}) est la récompense future attendue pour effectuer l'action (a_t) à partir de l'état actuel (s_t). L'agent dans le QL apprend à calculer progressivement la récompense attendue dans l'environnement donné à travers plusieurs itérations en utilisant les Différences Temporelles (TD). Les récompenses calculées sont enregistrées dans un tableau en même temps que l'état et les actions tout au long du modèle. Le processus QL commence par l'initialisation d'une Q-table avec des Q-values aléatoires. Une action aléatoire est choisie et effectuée à partir de l'état actuel. A partir de la récompense obtenue, la Q-value [32] pour cette paire état-action est mise à jour comme suit :

$$Q^{*}(s_t, a_t) = Q(s_t, a_t) + \alpha(R_t + \gamma.max_a(Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{1.2}$$

dont $Q^{*}(s_t, a_t)$ désigne la Q-value mise à jour basée sur la récompense obtenue (R_t) tout en effectuant une action (a_t) à partir de l'état (s_t). α représente le taux d'apprentissage(learning rate) , et γ est le facteur d'acceptation(discount rate).

Au début de l'exploration, les actions sont prises au hasard et la Q-table ne peut pas être précise. Après plusieurs itérations, cette Q-table sera mise à jour pour bien apprendre l'environnement [33] . Le pseudo-code pour concevoir le Q-learning est présenté dans l'Algorithme 1. De cette façon, l'agent augmente sa confiance pour identifier l'action correcte à partir de l'état actuel dans n'importe quel environnement complexe à travers l'exploration et l'exploitation. La possibilité de corriger le modèle en fonction de l'environnement de façon à atteindre les états souhaités rend Q Learning plus populaire pour diverses applications.

Algorithm 1 Q-Learning

1: Initialize Q-table with random values for all state-action pairs.
2: For each episode :
3: Initialize the starting state s_t.
4: Repeat until the current state is a terminal state :
5: choose a greedy action (a_t)
6: perform action, (a_t) and calculate reward, (r_t)
7: Update Q value based on Eq.1.1
8: Move to next state (s_{t+1})
9: End the episode.
10: Repeat Step 2 for the desired number of episodes or until convergence.

1.3 Conclusion

Ce chapitre a donné un aperçu des scénarios de déchargement de tâches dans le Edge computing. Dans un premier temps, les concepts du cloud et edge computing ont été développés, ainsi que leurs avantages. Ensuite, la nécessité du déchargement de tâches et les défis liés à sa gestion ont été expliqués. Une nouvelle stratégie de déchargement visant à surmonter les limites existantes, sera proposée en détail dans les chapitres à venir.

Chapitre 2

Formulation du problème

2.1 Introduction

Dans ce chapitre, la Section 2.2 présente en detail le modèle du système détaillé, et la Section 2.3 formule la fonction objective pour les scénarios de déchargement de tâches dans un environnement edge cloud collaboratif. La liste des notations est donnée dans le tableau 2.1.

Notation	Description
x_1^t,x_2^t,x_3^t	Déchargement d'une tâche t respectivement en local, en serveur Edge, en serveur cloud.
D_ij^e,B_ij^e	le débit de données et la bande passante entre l'appareil i et le serveur Edge j.
D_j^r,B_j^r	le débit de données et la bande passante entre le serveur Edge j et le serveur cloud.
P_j^i	Puissance du signal entre l'appareil i et le serveur Edge j.
I_i	Interférence à l'appareil i à partir des autres serveurs.
t_i	Puissance de transmission d'un appareil générant une tâche i.
d_ij	La distance entre un appareil i et un serveur Edge j.
l_t	La longueur d'une tâche (en Millions d'instructions MI).
f_i^l	Le taux de traitement d'un appareil (MI/sec).
S_t	La taille d'entrée des données.
y_{jt}	La décision de mise en cache d'une tâche (k_t) dans un serveur j.
f_j^e	Le taux de traitement d'un serveur Edge j(MI/sec).
f^r	le taux de traitement du serveur cloud.
α	Les pertes de transmission.
ρ_t	Le delai autorisé pour une tâche.
τ	Le nombre de demandes pour une tâche i.
R	Le nombre total des tâches.
c_j	La taille totale de la mise en cache d'un serveur j.
c_{avj}	La taille de mise en cache disponible d'un serveur j.
σ_j	Le temps d'accès à la mémoire d'un serveur j.
h_{ij}	le gain de canal entre un appareil i et un serveur j.

TABLE 2.1 – Liste des notations

2.2 Modélisation du système

La modélisation du système considéré suppose que des tâches sont générées périodiquement à partir des appareils utilisateurs tels que des smartphones, des ordinateurs portables, des capteurs et des Raspberry Pi. Les paramètres de tâche comprennent le délai autorisé, la longueur (en millions d'instructions), la taille d'entrée, la taille des fichiers de dépendance et la taille de sortie pour exécuter des tâches de type réalité augmentée, infotainment et e-santé. Le système comprend un serveur cloud central, un serveur edge et plusieurs machines virtuelles sur le cloud et le serveur edge.

Un serveur cloud central [34] est un centre de données doté d'un grand espace de stockage et d'une grande capacité de traitement pour gérer des données massives à tout moment. On considère un serveur cloud centralisé (R) qui peut gérer l'ensemble du réseau. Avec le cloud, un ensemble de serveurs edge E = e_0,e_1,...,e_s est pris en compte. Les serveurs edge [6] sont de petits centres de données plus proches des appareils utilisateurs avec une capacité de stockage et de traitement limitée pour traiter les données générées par les applications IoT. À tout instant, si le serveur edge associé à un appareil utilisateur est confronté à une interruption de service, un autre serveur edge situé à proximité de l'utilisateur peut être utilisé pour le traitement qu'on appelle un serveur edge adjacent.

De plus, on considère aussi des serveurs cloud et edge distants qui hébergent plusieurs machines virtuelles pour permettre l'exécution parallèle de tâches. Une machine virtuelle [35] est une ressource logicielle qui peut déployer et traiter des données au lieu d'utiliser un ordinateur physique. Elle contient une unité centrale de traitement (CPU) et un disque mémoire pour stocker et exécuter des fichiers. Plusieurs machines virtuelles peuvent être hébergées par un seul ordinateur hôte pour permettre l'exécution parallèle. Le nombre de machines virtuelles disponibles sur le cloud central est bien plus important que sur le serveur edge pour reproduire une situation réelle. Les capacités de traitement diffèrent également considérablement en fonction du nombre de machines virtuelles. La tâche générée peut être exécutée localement, sur le serveur edge ou sur le cloud distant à tout instant [36] , comme indiqué ci-dessous :

$$x_l^t + x_e^t + x_c^t = 1 \tag{2.1}$$

où x_l^t,x_e^t,$x_c^t \in \{0,1\}$ désignent respectivement la décision de déchargement, sur l'appareil utilisateur, sur le serveur edge ou sur le cloud distant.

On suppose que les serveurs edge sont placés avec les stations de base pour modéliser le délai de service. Un serveur edge de reporting est associé à chaque appareil utilisateur en fonction de leur distance. Ce dernier est associé si la distance calculée est inférieure au rayon de couverture du serveur edge. À tout moment, s'il se trouve dans la zone de couverture de plusieurs serveurs edge, alors le débit de données est utilisé pour associer un serveur edge. Le débit de données

sans fil [12] entre le serveur edge et l'appareil utilisateur peut être calculé à l'aide de la puissance de transmission, de la distance, de l'interférence des autres stations de base, comme indiqué ci-dessous :

$$D^e_{ij} = B^e_{ij} log_2(1 + \frac{p^i_j}{\sigma^2 + I_i}) \tag{2.2}$$

Où D^e_{ij} est le débit de données entre un appareil i et un serveur de edge j, B_{ij} désigne la bande passante du canal entre l'appareil i et le serveur edge j, p^i_j désigne la puissance du signal du dispositif i au serveur edge j, σ^2 désigne le bruit du signal, et I_i désigne l'interférence d'autres serveurs edge. La puissance du signal d'un dispositif peut être calculée en utilisant le gain de canal (h_{ij}), la puissance de transmission (t_i), la distance entre le dispositif et le serveur edge (d_{ij}), et les pertes du trajet (α) comme suit :

$$p^i_j = t_i h_{ij} (d_{ij})^{-\alpha} \tag{2.3}$$

2.2.1 Modélisation de la mise en cache

Les données fréquemment utilisées peuvent être stockées en mémoire interne ou en RAM en fonction des exigences de temps d'accès connues sous le nom de mise en cache. Les données envoyées au cloud distant peuvent être cachées localement sous de nombreuses communications avec le cloud central représentent une charge considérable sur la bande passante et augmentent le délai de traitement des entrées et des résultats. Contrairement au cloud centralisé, les serveurs edge ont une mémoire limitée pour stocker les données et ont besoin d'une utilisation optimale pour la gestion des ressources.

Mise en cache basée sur les conteneurs : Le paquet logiciel de dépendances et de bibliothèques nécessaires à l'exécution d'une tâche de type application est un conteneur. Ces conteneurs doivent être téléchargés chaque fois par un serveur edge ou un appareil utilisateur pour exécuter une tâche de type application. Mettre ces conteneurs en cache localement ou sur le serveur edge en fonction de la disponibilité du stockage et de la fréquence des demandes d'application élimine le téléchargement récurrent des dépendances. En outre, l'espace de stockage limité peut être géré en remplaçant les conteneurs en cache existants par des conteneurs populaires mis à jour d'un type d'application. Comme la popularité d'une application varie en fonction de l'emplacement, différents serveurs edge peuvent avoir des conteneurs mis en cache différents, améliorant ainsi l'utilisation de la bande passante.

Par conséquent, il est nécessaire de faire un choix approprié pour décider si les données doivent être mises en cache sur un serveur edge ou sur un appareil utilisateur. La décision de stocker doit être prise en considérant attentivement le bénéfice de la mise en cache d'un conteneur et le coût de stockage. Le bénéfice de la mise en cache d'un conteneur [25] peut être calculé en fonction du

nombre de demandes pour une application (τ) et du nombre total de demandes (R) sur une période donnée.

$$P_t = \frac{\tau}{\rho_t R} log \frac{c_{av_j} f_j^e \sigma_j}{c_j} \tag{2.4}$$

Où (P_t) designe le bénéfice de cache d'une application (k_t), ρ_t est la latence autorisée, c_{av_j} représente la taille de cache disponible sur le serveur edge j, f_j^e est la vitesse de calcul du serveur edge j, σ_j est le temps d'accès à la mémoire du serveur edge j, c_j est la taille totale de cache du serveur edge j. La perte de données en raison du stockage dans le cache peut être exprimée par l'équation 2.5, où β représente le coût unitaire de stockage pour le cache des données d'entrée s_t, et x_e^t représentent la décision de décharger la tâche au serveur edge.

$$C_l = x_e^t s_t \beta - \frac{\tau}{\rho_t R} log \frac{c_{av_j} f_j^e \sigma_j}{c_j} \tag{2.5}$$

2.2.2 Scénarios de déchargement

À tout moment, un appareil peut exécuter une tâche dans trois emplacements : localement, sur un serveur edge, dans le cloud distant. Dans le cas où le appareil utilisateur est un capteur, les données générées doivent être traitées à l'extérieur car il n'a pas la capacité de les traiter localement. En plus de l'emplacement du déchargement, une machine virtuelle parmi les machines disponibles doit également être sélectionnée en fonction du temps d'attente si la tâche est déchargée vers le serveur edge ou le cloud distant. À tout moment, la consommation d'énergie est due au téléchargement des conteneurs liés aux tâches depuis le cloud distant et à l'exécution de la tâche. Si une tâche doit s'exécuter localement, alors les conteneurs sont d'abord téléchargés sur le serveur edge et ensuite envoyés au dispositif utilisateur. De même, la latence encourue dans le traitement des conteneurs doit également être prise en compte. Si les tâches sont exécutées sur le serveur edge ou dans le cloud distant, alors le délai d'envoi des paramètres liés à la tâche est également pris en compte.

2.2.2.1 Déchargement local

Si la tâche est déchargée localement, alors les ressources de traitement de l'appareil sont utilisées pour réaliser la tâche, et le délai est le temps consommé pour l'exécuter. Le traitement local des tâches peut prendre plus de temps car la capacité de traitement de l'appareil est limitée. Si la tâche est générée à partir d'un appareil tel qu'un capteur, il n'aura même pas la possibilité de traiter localement, et la tâche doit être déchargée vers un emplacement différent. La consommation d'énergie locale [36] comprend le téléchargement des conteneurs d'une tâche depuis le cloud distant via le serveur edge ainsi que son traitement. À tout moment, si la tâche k_t de longueur l_t est

traitée localement, la consommation d'énergie peut être calculée comme suit :

$$E_t^l = kl_t f_i^{l^2} + (1 - \gamma_{jt})(\frac{t_i c_t}{D_{ij}^e} + \frac{t_i c_t}{D_j^r}) \tag{2.6}$$

Où k est la capacitance de commutation, f_i^l represente le taux de traitement d'un appareil mesuré en Millions d'Instructions par seconde, l_t est la longueur d'une tâche son unité est Millions d'instructions (MI), c_t représente la taille du conteneur, D_{ij}^e désigne le taux de transmission de données entre un dispositif i et un serveur Edge j, y_{jt} représente la décision de mise en cache, et D_j^r désigne le taux de transmisison de données entre le serveur edge j et le serveur cloud. Pour exécuter une tâche localement, les conteneurs doivent être téléchargés depuis le cloud distant via le serveur edge s'ils ne sont pas mis en cache. Par conséquent, l'équation 2.6 comprend l'énergie nécessaire pour télécharger les conteneurs et pour traiter les tâches localement. l'équation de délai est modélisée pour calculer le temps nécessaire pour traiter la tâche localement ($\frac{l_t}{f_i^l}$) et le temps nécessaire pour charger le conteneur (c_t) depuis le cloud via le serveur edge si la tâche n'est pas en cache (y_{jt}). Le débit de données entre le serveur edge j et le dispositif utilisateur i est représenté par D_{ij}^e. De même, le débit de données entre le serveur cloud et le serveur edge j est représenté par D_j^r. Par conséquent, le temps nécessaire (latence) [37] pour exécuter une tâche localement, y compris le délai de téléchargement du conteneur depuis le serveur cloud , peut être exprimé comme suit :

$$L_t^l = (\frac{l_t}{f_i^l}) + (\frac{c_t}{D_{ij}^e} + \frac{c_t}{D_j^r})(1 - \gamma_{jt}) \tag{2.7}$$

2.2.2.2 Déchargement sur le serveur Edge

Si un appareil utilisateur décide de décharger une tâche (k_t) vers le serveur edge alors ses machines virtuelles sont utilisées pour traiter la tâche. Dans ce scénario, le temps nécessaire pour envoyer la tâche vers le serveur edge, ainsi que le temps de téléchargement du conteneur, sont également ajoutés à la latence. Étant donné que la taille du résultat est beaucoup plus petite que la taille de la demande, le délai subi pour renvoyer les résultats est ignoré. Il peut y avoir plusieurs machines virtuelles disponibles sur le serveur edge qui exécutent des tâches en parallèle, et les tâches seront déchargées vers une machine virtuelle qui a le moins de tâches en attente dans la file d'attente. Le taux de traitement de la machine virtuelle sélectionnée est représenté par f_i^l, et la taille de la tâche est représentée par s_t. Au niveau du serveur edge, la consommation d'énergie [36] comprend le téléchargement du conteneur (c_t) depuis le cloud distant si la tâche n'est pas mise en cache (γ_{jt}), le téléchargement des données (s_t) depuis l'appareil utilisateur, puis l'exécution de la tâche. En fonction de la décision de mise en cache d'une tâche, (γ_{jt}), la consommation totale

d'énergie est calculée par :

$$E_t^e = \frac{t_i s_t}{D_{ij}^e} + \frac{t_i c_t}{D_j^r}(1 - \gamma_{jt}) + k l_t {f_i^e}^2 \tag{2.8}$$

où t_i est la puissance de transmission de l'appareil utilisateur et f_i^e est la capacité de traitement du serveur edge j. En ignorant le temps nécessaire pour renvoyer les résultats à l'appareil, le delai de traitement comprend trois composantes : l'envoi de la tâche au serveur edge (s_t), le chargement du conteneur (c_t) depuis le cloud distant si la tâche n'est pas mise en cache (γ_{jt}), et le traitement de la tâche. Par conséquent, la latence [37] pour décharger une tâche (k_t) vers le serveur edge peut être représentée par :

$$L_t^e = \frac{s_t}{D_{ij}^e} + \frac{c_t}{D_j^r}(1 - \gamma_{jt}) + \frac{l_t}{f_j^e} \tag{2.9}$$

D_{ij}^e désigne le débit de transmission de données entre le dispositif i et le serveur edge j, D_j^r désigne le débit de données entre le serveur edge j et le cloud distant, et l_t est la longueur de la tâche (k_t).

2.2.2.3 Déchargement sur le serveur Cloud

Si la tâche est déchargée vers le cloud distant, le retard de transport (Backhaul delay)[38] pour envoyer la tâche de la station de base au cloud distant sera ajouté, ainsi que la latence rencontrée pour l'envoyer à la station de base (co-localisée avec le serveur dgee). Par conséquent, la consommation d'énergie et la latence pour traiter la tâche au cloud distant incluent le chargement des données d'entrée (s_t) depuis le dispositif i via le serveur edge j, puis l'exécution de la tâche. Si D_j^r est le débit de données entre le serveur edge j et le cloud distant, D_{ij}^e désigne le débit de données entre l'appareil i et le serveur edge j, et f_j^r est le taux de traitement du cloud distant, alors la consommation d'énergie et la latence pour décharger une tâche (k_t) vers le cloud distant peuvent être calculées par :

$$E_t^r = \frac{t_i s_t}{D_{ij}^e} + \frac{t_i s_t}{D_j^r} + k l_t {f_i^r}^2 \tag{2.10}$$

$$L_t^r = \frac{s_t}{D_{ij}^e} + \frac{s_t}{D_j^r} + \frac{l_t}{f_j^r} \tag{2.11}$$

2.3 Fonction objective et contraintes

L'objectif principal du déchargement et de l'allocation des ressources dans l'environnement de cloud edge hétérogène est de minimiser la consommation d'énergie totale tout en garantissant que les tâches sont traitées dans les délais autorisés. Par conséquent, la fonction objective comprend à la fois la consommation d'énergie et la latence en tenant compte des contraintes des ressources de

traitement.

$$P_t : min \sum_t (w_1 L + w_2 E + w_3 C_l) \tag{2.12}$$

$$L = x_l^t L_t^l + x_e^t L_t^e + x_c^t L_t^r \tag{2.13}$$

$$E = x_l^t E_t^l + x_e^t E_t^e + x_c^t E_c^r \tag{2.14}$$

$$C_l = x_e^t s_t \beta - \frac{\tau}{\rho_t^R} log \frac{c_{av_j} f_j^e \sigma_j}{c_j} \tag{2.15}$$

L désigne la latence, E est la consomation d'energie, C_l est la perte de mise en cache pour le traitement de la tâche (k_t).

w_1, w_2 et w_3 representent respectivement les poids de la latence, de la consomation d'energie et de la perte de mise en cache. Ces poids peuvent être mis à jour en fonction des exigences de l'utilisateur.x_l^t, x_e^t et x_c^t représentent respectivement la décision de déchargement pour effectuer le traitement localement, sur le serveur edge et sur le cloud. La fonction objective (P_t) doit être minimisée de telle manière que :

$$x_l^t + x_e^t + x_c^t = 1. \tag{2.16a}$$

$$x_l^t \in \{0, 1\}. \tag{2.16b}$$

$$x_e^t \in \{0, 1\}. \tag{2.16c}$$

$$x_c^t \in \{0, 1\}. \tag{2.16d}$$

$$x_e^t s_t < D. \tag{2.16e}$$

La contrainte (2.16a) montre qu'une tâche peut être déchargée dans un seul endroit (localement, sur le edge ou le cloud distant). La contrainte (2.16e) indique que la taille des données de la tâche en cours d'exécution sur un serveur edge doit être inférieure à sa capacité de stockage. Par conséquent, nous exploitons Q Reinforcement Learning (QRL) pour résoudre la fonction objective comprenant l'énergie et la latence. La section 3 explique l'importance du QRL [35] et son application au problème P_t.

2.4 Conclusion

Ce chapitre présente un modèle de système composé d'appareils utilisateurs statiques et mobiles générant des tâches de types d'applications réalistes avec des serveurs edge et un cloud central distant. Le modèle de communication entre les différents centre de données est également expliqué à l'aide d'équations mathématiques. Une nouvelle mise en cache basée sur les conteneurs(container-based caching) pour stocker les dépendances des applications fréquemment utilisées est proposée. Ensuite, l'énergie consommée, et la latence encourue pour traiter les tâches générées localement,

au serveur edge, et au cloud distant sont élaborés. Enfin, la fonction objective de la consommation d'énergie, de la latence et du coût de mise en cache est formulée sous différentes contraintes.

Chapitre 3

Méthodologie proposée

3.1 Introduction

Ce chapitre présente le principe de mise en cache et met en évidence sa pertinence dans le scénario de déchargement. la section 3.3 détaille l'effet du choix de l'emplacement de déploiement pour un réseau. Dans la Section 3.6, le Q Reinforcement Learning (QRL) est proposé pour résoudre la fonction objective formulée au Chapitre 2. En outre, les algorithmes proposés sont présentés en détail avec l'analyse théorique. Enfin, une comparaison avec des algorithmes de référence (Benchmark algorithms) pour valider l'importance de l'approche proposée est également introduite.

3.2 Mise en cache basée sur les conteneurs

Les fichiers de configuration d'installation d'application fréquemment utilisés, qui peuvent être mis en cache dans le stockage interne ou la mémoire vive (RAM), sont définis comme étant basés sur le cache en fonction des besoins en temps d'accès. Le regroupement logiciel des dépendances et des bibliothèques requises pour exécuter une tâche de type application est appelé conteneur. Ces conteneurs doivent être téléchargés à chaque fois par un serveur edge ou un appareil utilisateur pour exécuter une tâche de type application. Par conséquent, le fait de transférer fréquemment des tâches vers un serveur cloud centralisé peut être évité en stockant localement des conteneurs populaires ou sur un serveur edge. De plus, le stockage limité disponible sur le serveur edge peut être utilisé en remplaçant de manière proactive les conteneurs populaires mis à jour par des versions obsolètes en fonction de la fréquence des demandes.

Ainsi, l'utilisation de la mise en cache basée sur les conteneurs(Container-based caching) permet de réduire la transmission de données et d'améliorer la qualité de livraison pour les utilisateurs. Cependant, il est nécessaire de développer une technique de déchargement optimale qui prend en compte les avantages de la mise en cache, la consommation d'énergie et la latence afin d'exécuter les tâches avec succès dans tous les environnements. Ce processus, appelé orchestration, consiste à

prendre des décisions sur les emplacements d'exécution des tâches en tenant compte de contraintes telles que la consommation d'énergie, la latence et la mise en cache. Un centre de données responsable de ce processus peut être désigné comme un orchestrateur. En plus de choisir un emplacement optimal pour l'exécution d'une tâche, il est essentiel de sélectionner l'emplacement de déploiement d'un algorithme. Ce dernier joue un rôle crucial dans l'obtention de performances réseau efficaces.

3.3 Analyse du déploiement de l'orchestrateur

L'emplacement du déploiement de l'orchestrateur joue un rôle important dans l'utilisation des ressources du réseau. Par exemple, il est coûteux en termes de bande passante et de délai d'envoyer fréquemment les informations des tâches générées à un orchestrateur centralisé. En plus des informations liées aux tâches, la disponibilité des ressources dans différents centres de données doit également être envoyée et les résultats doivent être transmis en fonction de l'intervalle de mise à jour du réseau. Cependant, une mise à jour fréquente des informations liées au réseau augmente la charge sur la bande passante disponible et peut entraîner une congestion du réseau.

Par conséquent, l'incorporation d'une technique de déchargement distribué peut améliorer les performances globales du système. Dans cette approche, chaque appareil utilisateur prend localement la décision d'exécution d'une tâche générée, réduisant ainsi les transferts fréquents de données liées aux tâches et d'informations sur la disponibilité des ressources locales et voisines. De plus, la bande passante disponible peut être utilisée pour envoyer les tâches sélectionnées nécessitant des serveurs edge et un traitement centralisé dans le cloud. Dans le scénario considéré, il est supposé que les paramètres du réseau sont mis à jour de manière globale. Par conséquent, chaque appareil utilisateur et centre de données dispose d'informations globales sur les ressources disponibles et la capacité de traitement.

3.4 Déchargement décentralisé

L'approche de déchargement décentralisé est une technique de déchargement distribué où tous les appareils utilisateurs décident localement l'emplacement de déchargement d'une tâche générée. L'algorithme est supposé être déployé localement, et les paramètres utilisés pour exécuter l'algorithme sont mis à jour de manière globale. Cette technique de déchargement distribué utilise les paramètres globaux du réseau tels que la bande passante disponible, les serveurs edge inactifs et leur puissance de traitement. Au lieu d'envoyer toutes les informations des utilisateurs avec les tâches générées vers un cloud centralisé, on maintient les informations globales sur le réseau ce qui permet de réduire la charge sur la bande passante disponible.

De plus, il surmonte le problème du point de défaillance unique dans le déchargement centralisé. Ce fonctionnement décentralisé réduit le transfert fréquent de données liées aux tâches et

d'informations sur la disponibilité des ressources locales et voisines. En outre, la bande passante disponible peut être utilisée pour envoyer des tâches sélectionnées qui nécessitent des serveurs edge et un traitement en cloud centralisé.

Dans la technique de déchargement distribué, l'algorithme sera exécuté localement pour tous les appareils utilisateurs et une décision de traiter la tâche sera prise, suivie de l'exécution de la tâche. On suppose qu'aucun surcoût supplémentaire n'est nécessaire pour exécuter l'algorithme localement et que le coût de stockage engendré par le déploiement de l'algorithme est négligeable par rapport aux accords de niveau de service stricts pour les applications IoT.

3.5 Modèle d'ordonnancement et de mise en cache des tâches basé sur l'approche "Online Q Reinforcement Learning"

La fonction objective (P_t) dans ce contexte ne peut pas être traitée à l'aide de la programmation linéaire en raison de la présence de diverses variables, telles que la bande passante disponible, la latence autorisée, l'emplacement de déchargement, la décision de mise en cache, l'espace de stockage et la capacité de traitement dans des centres de données à plusieurs niveaux (local, serveur edge et cloud distant). Ces variables introduisent des complexités et des non-linéarités qui ne peuvent pas être facilement capturées ou optimisées à l'aide de techniques de programmation linéaire. De plus, la séquence d'actions effectuées pour les tâches augmente encore le nombre de solutions possibles à tout moment. Par conséquent, le Online Q Reinforcement Learning est utilisé pour résoudre ce défi.

Le problème P_t Eq :(2.12) doit d'abord être transformé dans le cadre du Q Reinforcement Learning pour appliquer l'algorithme. Un aperçu du modèle de système proposé est présenté dans la figure 3.1.

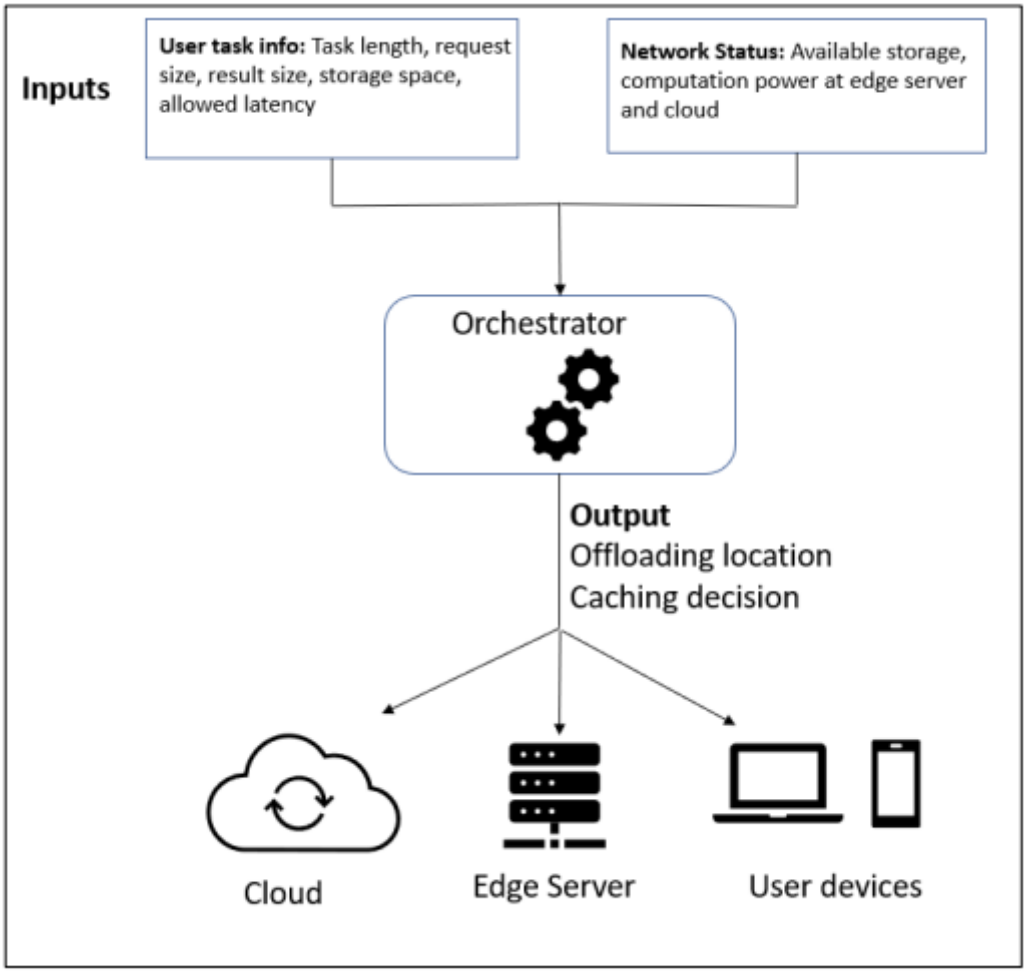

FIGURE 3.1 – Aperçu du modèle proposé

Si l'emplacement de déchargement choisi est en serveur Edge ou en cloud distant, une machine virtuelle est sélectionnée pour exécuter une tâche en fonction du temps d'attente le plus court afin d'éviter la concurrence. Les principaux composants du QRL comprennent l'État, l'Action, la Récompense et l'État Suivant, qui peuvent être définis comme suit :

a) **Etat** : Ce sont les paramètres utilisés pour définir le modèle du système. L'état représente les ressources disponibles pour le traitement et les paramètres associés à la tâche dans le problème de planification des tâches. En se basant sur le modèle du système présenté dans la Section 2, l'état pour n'importe quelle tâche (k_t) peut être exprimé comme suit :

$$State, s_t = \{l_t, \rho_t, s_t, f_i^l, f_j^e, f_j^r\} \tag{3.1}$$

où l_t représente la longueur de la tâche, ρ_t est la latence autorisée, s_t est la taille d'entrée des données de la tâche (k_t), f_i^l est le taux de traitement local, f_j^e est le taux de traitement au niveau du serveur Edge, f_j^r est le taux de traitement au niveau du serveur cloud pour la tâche (k_t).

b) **Action** : En se basant sur l'état actuel (s_t), l'agent sélectionnera une action à effectuer. Dans notre modèle de système, l'action comprend la décision de déchargement et de mise en cache, comme illustré ci-dessous :

$$Action, a_t = \{x_1^t, x_2^t, x_3^t, \gamma_{jt}\} \tag{3.2}$$

où x_1^t,x_2^t,x_3^t représentent respectivement le déchargement local, sur le serveur Edge et sur le serveur Cloud. γ_{jt} désigne la décision de mise en cache au niveau du serveur Edge.

c) **Recompense** : notre objectif est de minimiser la fonction coût donnée, P_t (Eq. (2.12)). Pour un état donné (s_t), l'action doit être sélectionnée de manière à maximiser la récompense. Par conséquent, la récompense pour l'application de QRL peut être exprimée en termes de fonction de coût, comme indiqué ci-dessous :

$$Reward, r_t = -P_t \tag{3.3}$$

où P_t est la fonction objective à minimiser.

d) **Etat suivant** : Il peut être décrit comme les ressources disponibles après qu'une tâche a été déchargée, ainsi que les paramètres de la tâche suivante qui attendent à l'orchestrateur.

$$NextState, s_{t+1} = \{l_{t+1}, \rho_{t+1}, s_{t+1}, f_{i+1}^l, f_{j+1}^e, f_{j+1}^r\} \tag{3.4}$$

3.5.1 Analyse théorique

Pour calculer la complexité dans l'environnement de simulation hétérogène donné, trois scénarios de déchargement sont pris en compte : l'exécution locale, sur le serveur edge et sur le cloud distant. De plus, il faut prendre en considération la décision de mise en cache d'une tâche. Par conséquent, la complexité [39] dans le traitement hors ligne de l'algorithme peut être exprimée comme O($3^n 2^n$), ce qui est très élevé et nécessite un traitement énorme pour obtenir de bonnes performances. En revanche, le QRL en ligne peut fournir une solution quasi-optimale avec une compléxité réduite à O((4)) et peut être mis en œuvre pour le traitement de nombreuses tâches.

Algorithm 2 Offloading decisions using QRL

1: Initialize number of user devices, simulation time, simulation area, WAN and LAN bandwidth, number of users, edge server coverage radius, and network update interval
2: for each simulation, do
3: Generate user devices, edge servers, remote cloud datacenters. Virtual machines are created for edge servers, remote cloud, and user devices based on their computing resources availability.
4: Random mobility pattern is assigned for all user devices.
5: Tasks are generated for all user devices, and a random time is assigned for scheduling. All tasks are scheduled to be dispatched at assigned time slots.
6: for each task, do
7: Find offloading location and caching decision using Algorithm 1 and send the task.
8: If offloading location is edge server or remote cloud,
9: find a *Virtual machine* (VM) at which the least number of tasks are waiting for execution.
10: if the task is not cached,
11: download the container
12: Execute the task and check if the task failed due to long delay, mobility, battery power. Increment tasks failed accordingly.
13: Send the results to the user device after validating the task failures.
14: Update energy consumption, CPU utilization, network model based on offloading location.
15: end for
16: end for

3.6 Stratégies d'ordonnancement de référence

Dans la littérature, on trouve plusieurs stratégies d'ordonnancement de référence. Pour valider notre approche, on a opté pour ces deux algorithmes comme indiqué ci-dessous :

a) **Round Robin** : Cette politique est utilisée dans de nombreux ordinateurs portables, tablettes et autres gadgets électroniques pour mettre en file d'attente plusieurs applications demandant des ressources de traitement. Chaque fois qu'une tâche arrive, cette politique itère à travers toutes les machines virtuelles disponibles pour en choisir une qui présente le temps d'attente le plus court pour l'exécution [40]. Étant donné que cette politique vérifie toutes les machines disponibles et que l'ordre de complexité est O(n), elle convient parfaitement aux applications à petite échelle uniquement. Néanmoins, cette politique ne prend pas en compte les spécifications de la tâche pour choisir un emplacement de déchargement.

b) **Trade-Off** : Il s'agit d'une politique spécialement conçue pour choisir l'emplacement d'exécution parmi le serveur local disponible, le serveur Edge et le cloud distant en attribuant des poids. Ces derniers sont à tous les emplacements de traitement disponibles en fonction du délai pour les atteindre et de l'énergie consommée. Ensuite, la politique choisit l'emplacement de déchargement avec le poids le plus faible et le délai d'exécution le plus court [40]. Cependant, cet algorithme ne prend en compte que les caractéristiques des serveurs Edge et du cloud centralisé pour décider de l'emplacement d'exécution de toute tâche.

De plus, les performances de l'algorithme proposé sont encore améliorées en incorporant une mise en cache basée sur les conteneurs. Les performances de la méthode suggérée peuvent être validées en mettant en œuvre une mise en cache basée sur les tâches, comme expliqué ci-dessous.

Task-based Caching : Pour toute tâche générée à partir d'une application, le stockage des données d'entrée et de sortie (résultats) peut être qualifié de mise en cache basée sur les tâches. Cela peut être bénéfique uniquement lorsque les utilisateurs interrogent ou demandent des données comparables. Cependant, il s'agit d'un scénario rare car différents utilisateurs peuvent utiliser une application similaire mais ne demandent pas les mêmes données. Par exemple, un bâtiment disposant d'une autorisation de reconnaissance faciale peut utiliser fréquemment une application de reconnaissance faciale, mais le visage d'entrée à comparer dans une base de données diffère souvent.

3.7 Architectures de déchargement

L'emplacement de déploiement de l'algorithme est également crucial pour les performances globales du système et l'utilisation des ressources. Pour cette raison, une technique de déchargement distribuée où tous les utilisateurs décident l'emplacement de déchargement d'une tâche générée localement est recommandée. L'impact du déploiement de l'orchestrateur est examiné à l'aide des politiques de référence suivantes :

a) **Déchargement centralisé** : Dans cette approche, le cloud centralisé détermine l'emplacement d'exécution de toutes les tâches générées dans un réseau. Cependant, le paradigme d'orchestration centralisée nécessite des informations globales sur les tâches, l'espace de stockage disponible, la puissance de traitement, le niveau de batterie des appareils utilisateurs, la bande passante du réseau pour exécuter périodiquement la politique de déchargement. Ces informations complètes doivent être communiquées fréquemment pour assurer une orchestration continue. Cependant, les mises à jour fréquentes des informations réseau augmentent la charge sur la bande passante disponible et retardent le traitement des tâches. De plus, les variations de disponibilité des ressources entre les intervalles de mise à jour du réseau seront ignorées et auront un impact sur le temps de traitement estimé des tâches.

b) **Déchargement sur le Edge** : Dans cette approche, les décisions d'exécution des tâches sont prises de manière décentralisée, c'est-à-dire que chaque nœud du edge peut prendre localement des décisions basées sur ses propres ressources disponibles, sans avoir à communiquer fréquemment avec un cloud centralisé. Cela réduit la charge sur la bande passante du réseau et permet une exécution plus rapide des tâches. De plus, le déchargement sur le edge tient compte des variations de disponibilité des ressources en temps réel. Les décisions d'exécution des tâches peuvent être ajustées en fonction de la disponibilité actuelle des ressources à proximité, ce qui permet d'optimiser leur utilisation et de réduire les délais de traitement.

3.8 Conclusion

Ce chapitre débute en présentant les techniques d'apprentissage par renforcement. Par la suite, les principales raisons justifiant son utilisation pour les scénarios de déchargement des tâches, accompagnées d'une analyse théorique. L'approche proposée d'apprentissage par renforcement en ligne est ensuite présentée en détail, avec une description mathématique. Des algorithmes sont également fournis pour illustrer comment le cadre proposé peut être appliqué pour atteindre l'objectif visé. Enfin, des politiques de référence(Benchmark Policies) sont introduites afin de valider les performances de l'algorithme décentralisé, en incluant la mise en cache des conteneurs. Dans le chapitre 4, l'approche recommandée sera examinée en la comparant à des algorithmes de référence grâce à des simulations approfondies.

Chapitre 4

Simulation et résultats

4.1 Introduction

Dans ce chapitre, nous utilisons PureEdgeSim [40], un outil basé sur Java, pour mener des expériences de simulation. PureEdgeSim est un simulateur d'événements discrets qui a la capacité d'exécuter des processus parallèles de gestion des ressources. Des services basés sur le cloud allant de (IaaS) jusqu'au (SaaS) peuvent être implémentés à l'aide de ce simulateur. L'approche proposée (Online QRL) est mise en œuvre dans PureEdgeSim à l'aide de Java et évaluée sur des profils réalistes d'applications IoT, puis comparée à différentes politiques de référence introduites dans la section 3.5.

4.2 Paramétrage de la simulation

Nous considérons un environnement de simulation comprenant un cloud centralisé, plusieurs serveurs Edge et des appareils utilisateurs tels que des smartphones, des ordinateurs portables, des capteurs et des Raspberry Pi pour évaluer l'approche proposée. Les spécifications des dispositifs et des centres de données sont répertoriées dans les tableaux 4.1- 4.3.

Type de l'appareil	***Million instructions per second* (MIPS)**	***Random Access Memory* (RAM) en (GB)**	**Stockage (en GB)**
Smart phone	25000	4	128
Raspberry Pi	16000	4	32
Laptop	110000	8	1024
Sensor	70000	4	0

TABLE 4.1 – Caractéristiques des appareils utilisateurs.

Caractéristiques du serveur Edge	Hôte 1		Hôte 2	
	Machine virtuelle 1	Machine virtuelle 2	Machine virtuelle 1	Machine virtuelle 2
MIPS (Million instructions/sec)	200000	200000	200000	200000
RAM (en GB)	4	4	4	4
Stockage (en GB)	20	20	20	20

TABLE 4.2 – Caractéristiques du serveur Edge.

Spécifications du cloud central	
Hôtes	2
Machine virtuelle/Hôte	8
MIPS pour chaque VM	250000
RAM(en GB)	16
Stockage (en GB)	20

TABLE 4.3 – Caractéristiques du cloud central.

En outre, les smartphones et les ordinateurs portables sont des appareils alimentés respectivement par batterie avec une capacité de 18,75 et 56,2 watt-heures. Le Raspberry Pi et les sensors *Internet of things* (IoT) sont des appareils filaires qui fonctionnent à l'aide d'une alimentation électrique. Les tâches seront générées après des intervalles inégaux et aléatoires à raison de 7 par minute simultanément à partir de plusieurs appareils utilisateurs. Les applications générées par ces appareils comprennent la réalité augmentée, la e-Santé, l'infotainment avec calcul intensif, ainsi que les types d'applications mentionnés dans le Tableau 4.4. La sensibilité des applications par rapport à la latence est également incluse dans le Tableau 4.4.
La bande passante disponible pour le réseau local sans fil WLAN et le réseau WAN est considérée comme étant de 300 Mégabits par seconde. La bande passante disponible (available bandwidth) sera mise à jour périodiquement en fonction du nombre de tâches utilisant le réseau. Le délai de propagation du WAN (propagation delay) est supposé être de 0,2 secondes. La vitesse de traitement (computation speed) d'un centre de données est mesurée en millions d'instructions par seconde MIPS. La superficie totale de simulation conçue est de 200x200 mètres pour placer les serveurs Edge, le cloud central et les appareils utilisateurs. Le rayon de couverture(coverage radius) dans lequel l'appareil utilisateur peut communiquer avec le serveur Edge est supposé être de 20 mètres, et le serveur de Edge peut couvrir un rayon de 200 mètres. La consommation d'énergie pour chaque bit transmis ou reçu, la dissipation d'énergie de l'amplificateur en espace libre, la dissipation d'énergie de l'amplificateur en propagation à chemins multiples sont considérées respectivement comme étant de $5x10^{-8} Joules/bit$, $1x10^{-11}\ Joules\ m^2/bit$, $13x10^{-16}\ Joules\ m^4/bit$.

Application	**Sensibilité à la latence(%)**	**Request size(en KB)**	**Result size(en KB)**	**Taille du conteneur(en KB)**	**Longueur de la tâche(en MI)**
Augmented Reality	98	1500	100	25000	60000
e-Health	5	10000	10000	13000	300000
Infotainment	98	50	50	9000	15000

TABLE 4.4 – Les types d'application.

Les appareils utilisateurs commenceront d'un point initial aléatoire à l'intérieur de la zone de simulation et se déplaceront à une vitesse de 1,4 mètres/seconde. Les ressources disponibles au niveau des serveurs de réseau à couches multiples, telles que l'espace de stockage, la puissance de la batterie, les MIPS et les paramètres réseau tels que la bande passante, seront mis à jour simultanément en fonction des décisions de déchargement. À l'exception du capteur, tous les autres dispositifs peuvent exécuter des tâches localement.

Une fois que la simulation démarre, des dispositifs utilisateurs comprenant un smartphone, un ordinateur portable, un Raspberry Pi et un capteur seront générés. Les types d'applications répertoriés dans le Tableau 4.4 seront programmés pour être générés à partir des dispositifs utilisateurs à un créneau horaire aléatoire dans le temps de simulation fourni. Une seconde aléatoire chaque minute est choisie pour planifier une tâche. Il est maintenu que 7 tâches sont générées par minute à partir de chaque dispositif. Notre algorithme QRL proposé trouve la décision de déchargement pour les tâches générées par différents dispositifs. Le modèle de simulation est exécuté pendant trois itérations, en commençant par 100 à 300 dispositifs, et le temps de simulation pour chaque itération est de 10 minutes. Le temps de simulation n'inclut pas le temps alloué à la génération des ressources nécessaires. QRL décidera la décision de déchargement pour toutes les tâches, et en fonction de la décision, la tâche est exécutée et les résultats sont renvoyés à l'utilisateur.

La puissance de transmission maximale de l'appareil utilisateur ,t_i, est fixée à 100 mW. Les modèles de gain de canal présentés dans la normalisation 3GPP [41] sont adoptés ici. Plus précisément, large scale fading du canal entre le serveur edge et l'appareil utilisateur est 140.7 + 36.7 log10(r), où r est en km. Le modèle de fading de Rayleigh est adopté pour le fading à petite échelle. La puissance du bruit(Signal noise) est σ^2 = 10^{-11} mW et le seuil d'interférence est I = -90 dBm. La bande passante du cloud et du edge est respectivement de 10 MHz et 5 MHz.

4.3 Résultats et analyse

Le QRL proposé est exécuté et on le compare à deux autres algorithmes de référence mentionnés dans la section 3.5 afin d'obtenir une comparaison équitable. Les tâches des types d'applications mentionnés dans le Tableau 4.4 sont générées à partir de plusieurs appareils d'utilisateurs statiques et mobiles, et sont déchargées en utilisant notre algorithme.

4.3.1 Évaluation des performances de l'approche "Online QRL"

On considère comme architectures de l'orchestrateur : (Cloud, edge et local) et (Edge et cloud). La figure 4.1 et 4.2 montre respectivement le nombre de tâches qui ont échoué à être exécutées sur (local,edge et serveur) et sur (edge et cloud) en raison de leur mobilité. À tout moment, si une tâche correspondant à un utilisateur est en cours d'exécution sur le serveur edge et que l'utilisateur sort de portée, l'exécution de la tâche est interrompue. La tâche doit être réexécutée dans ce cas et le temps de traitement augmente. Si la latence accrue ne parvient pas à satisfaire les exigences, il est considéré que la tâche n'a pas été exécutée à temps en raison de la mobilité, et on peut observer que le QRL a surpassé les autres algorithmes. La figure 4.3,4.4 montrent le nombre de tâches qui ont échoué à être exécutées en raison du retard.

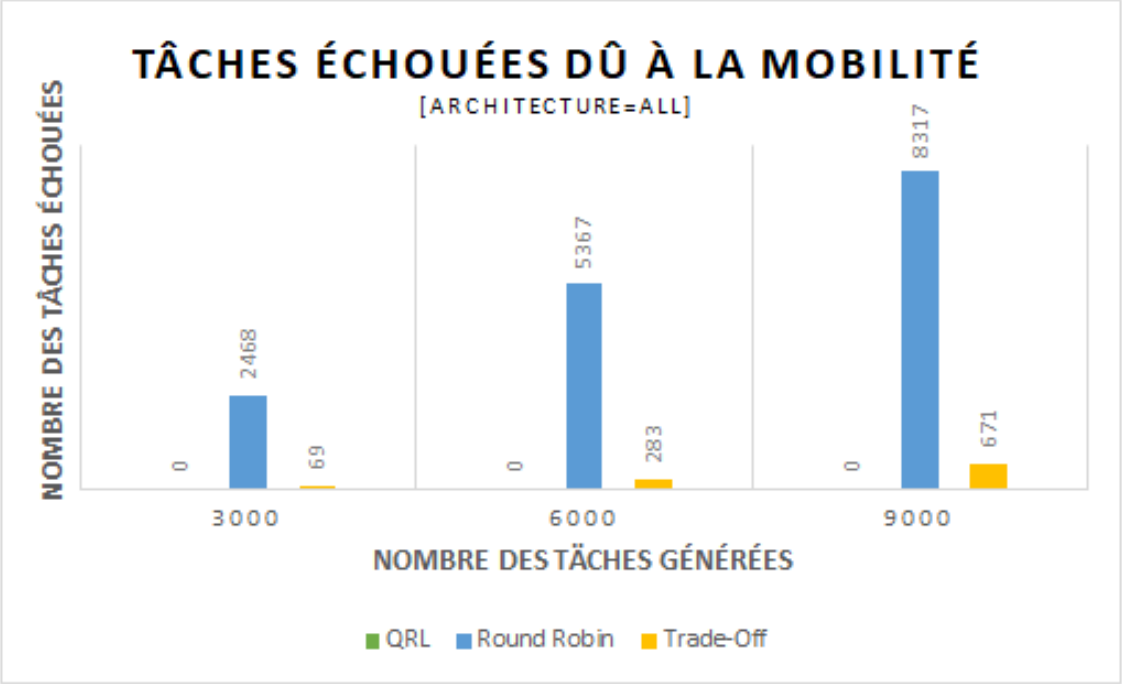

FIGURE 4.1 – Tâches échouées dû à la mobilité [ARCHITECTURE=ALL]

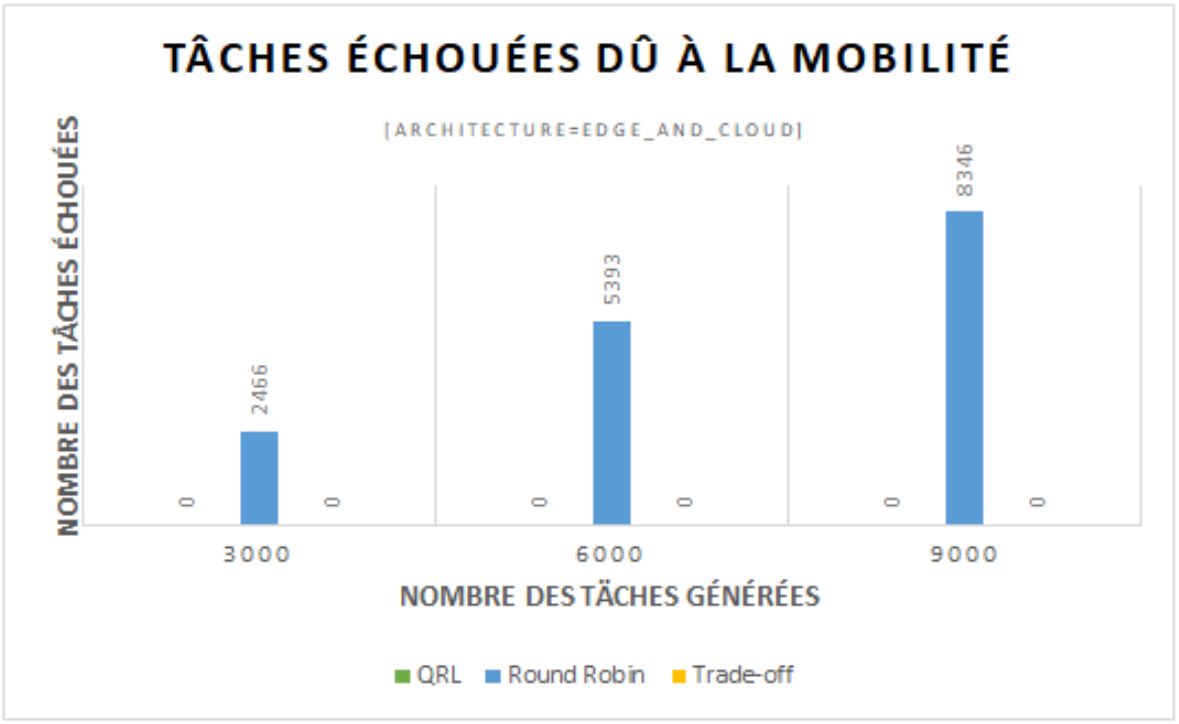

FIGURE 4.2 – Tâches échouées dû à la mobilité [ARCHITECTURE= EDGE_and_CLOUD]

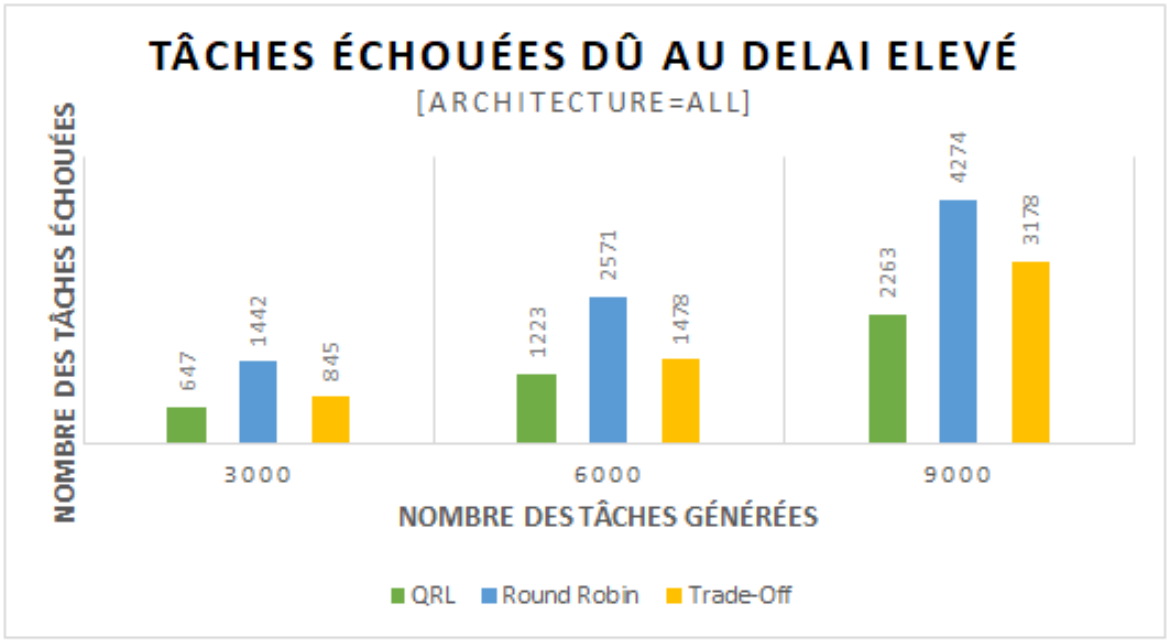

FIGURE 4.3 – Tâches échouées dû au long délai [ARCHITECTURE=ALL]

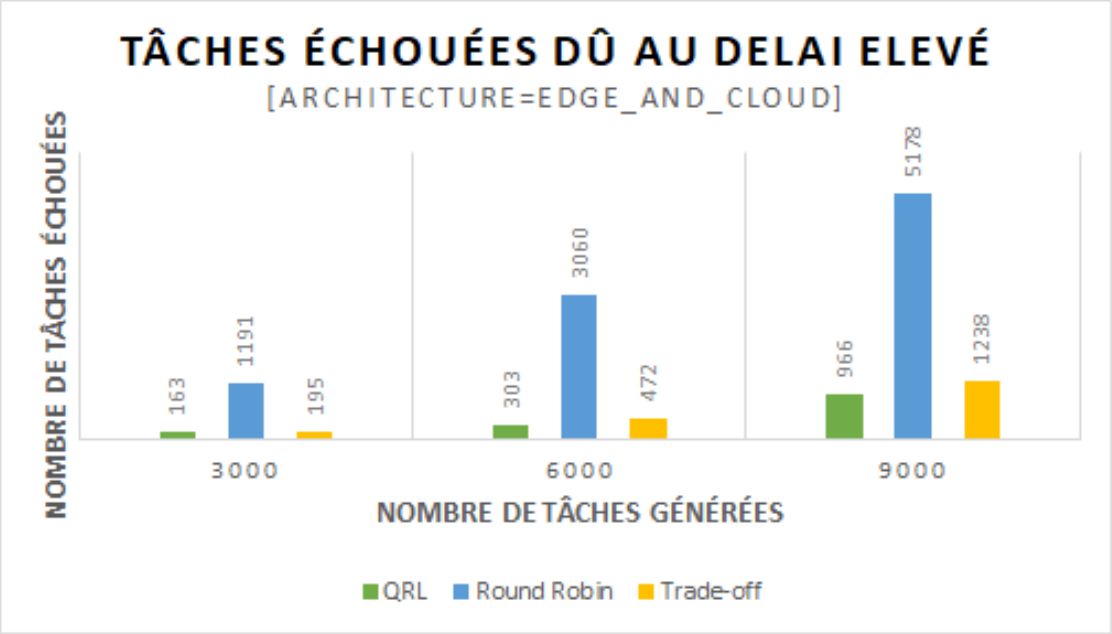

FIGURE 4.4 – Tâches échouées dû au long délai [ARCHITECTURE= EDGE_and_CLOUD]

On peut également observer que QRL a surpassé les autres algorithmes et a réduit les échecs de tâches comme le montre la Figure 4.3. Les performances insatisfaisantes de Round Robin et Trade-off peuvent être attribuées à leur complexité pour trouver un emplacement de déchargement. Ils itèrent à travers l'ensemble de la liste des ressources, y compris les appareils des utilisateurs, pour décider d'un emplacement de déchargement.
La figure 4.5,4.6 montrent le délai moyen d'exécution des tâches selon différents algorithmes. Le délai moyen d'exécution correspond au temps nécessaire pour calculer la tâche sur une machine virtuelle, et nous pouvons constater que l'algorithme Trade-Off a surpassé QRL pour plusieurs itérations. Cela s'explique par le fait que l'algorithme Trade-Off itère à travers l'ensemble de la liste des machines virtuelles et trouve une machine virtuelle de déchargement avec le plus faible délai d'exécution. L'itération à travers l'ensemble de la liste pour chaque tâche augmente la complexité de l'algorithme et nécessite plus de temps pour trouver une décision de déchargement à mesure que le nombre d'appareils augmente. Par conséquent, la fonction objective Trade-Off minimise le délai d'exécution et ne tient pas compte de la consommation d'énergie. Étant donné que les algorithmes Round Robin et Trade-Off parcourent toute la liste et choisissent successivement un emplacement, ils sont plus adaptés pour valider les performances de QRL.

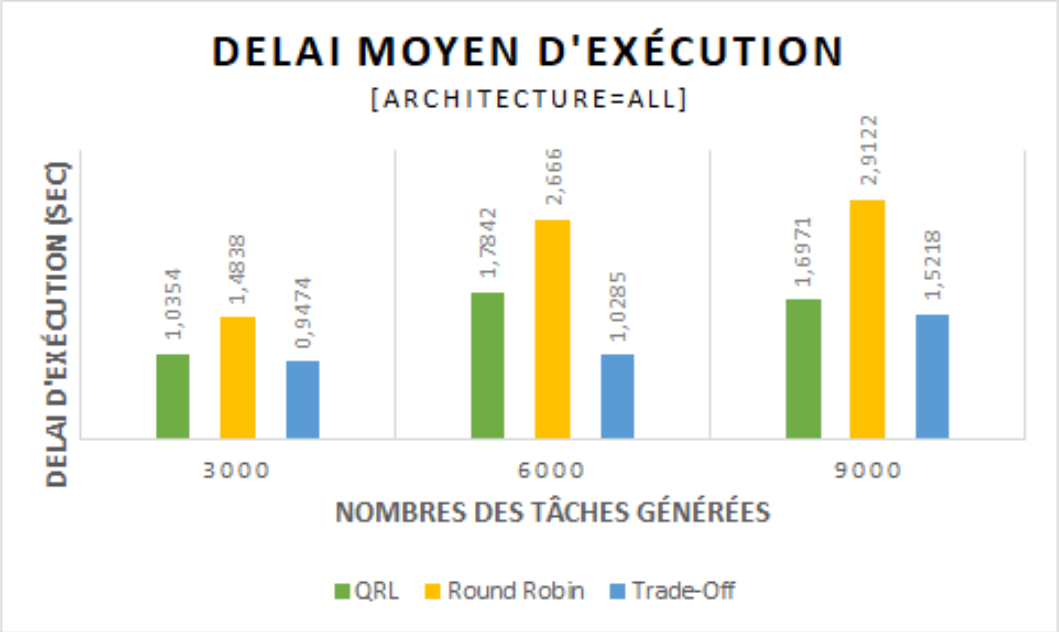

FIGURE 4.5 – Délai moyen d'éxecution [ARCHITECTURE=ALL]

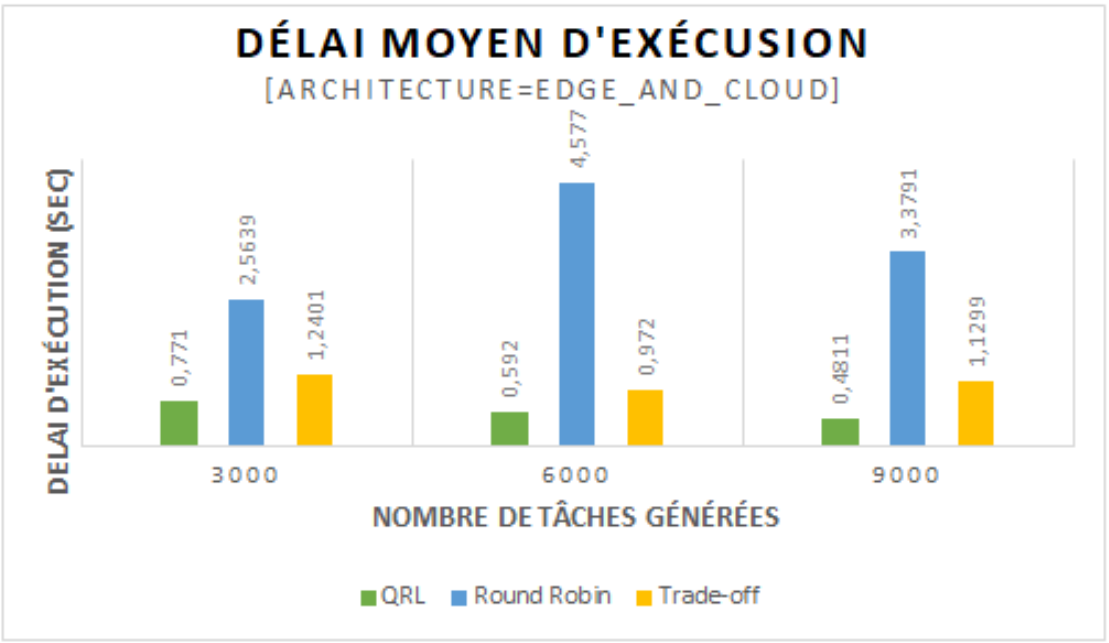

FIGURE 4.6 – Délai moyen d'éxecution [ARCHITECTURE= EDGE_and_CLOUD]

La gestion efficace des ressources disponibles dans les centres de données se reflète dans les figures 4.7 et 4.8, qui représentent la consommation énergétique moyenne.

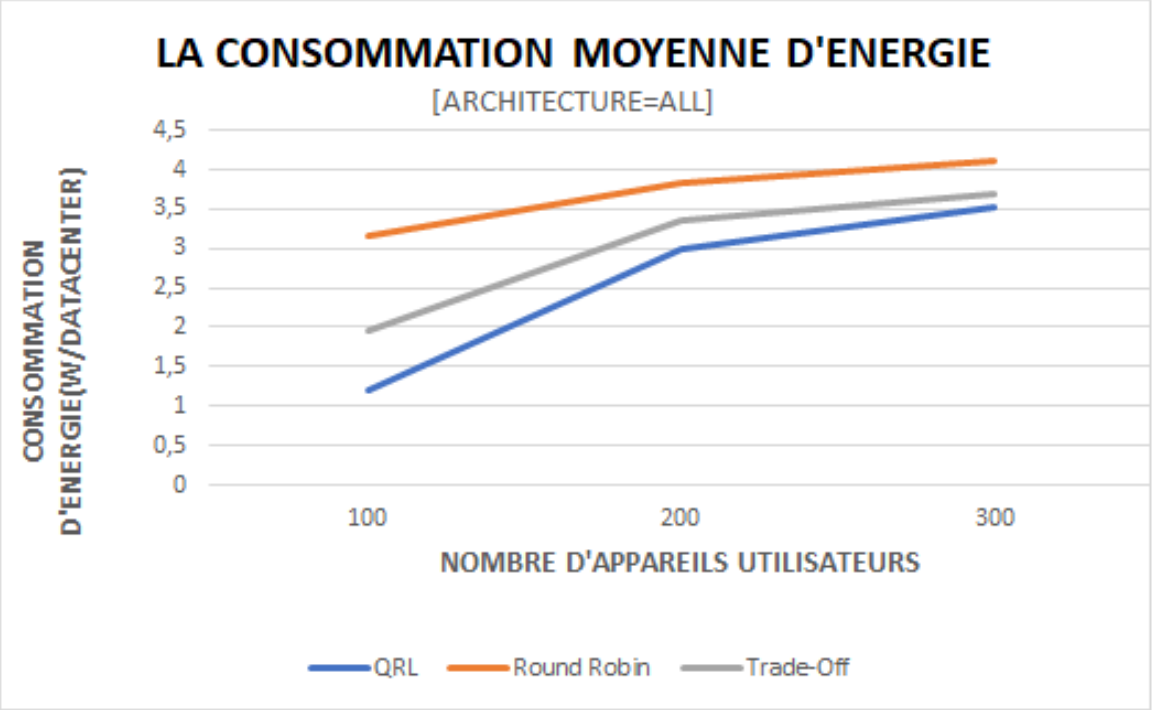

FIGURE 4.7 – La consommation d'énergie moyenne [ARCHITECTURE=ALL]

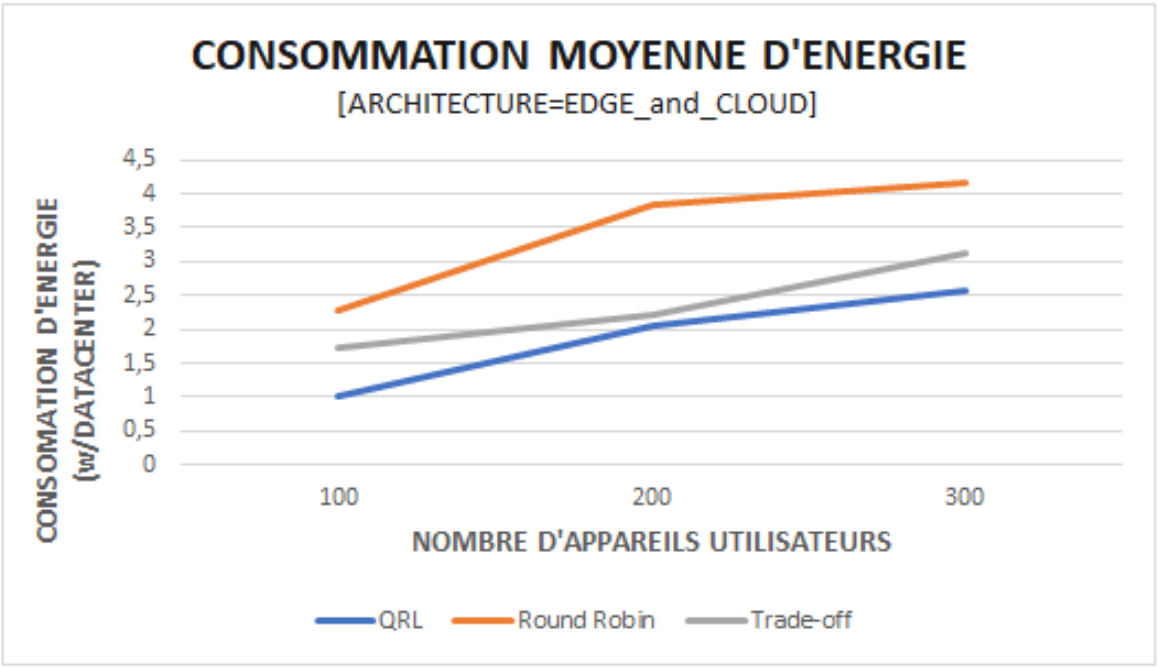

FIGURE 4.8 – La consommation d'énergie moyenne [ARCHITECTURE= EDGE_and_CLOUD]

Le QRL en ligne proposé surpasse le Round-Robin et le Trade-off en optimisant la consommation énergétique lors de la décision de l'emplacement de déchargement. Étant donné que l'algorithme proposé vise à minimiser la consommation d'énergie des appareils utilisateurs, il est important que

les appareils ne se retrouvent pas à court de batterie en raison de traitement local. La figure 4.9 et 4.10 montrent la puissance résiduelle de l'appareil après la fin de la simulation. On peut observer que le QRL exécute les tâches qui nécessitent le moins de ressources de traitement en local. Par conséquent, la consommation de la batterie est négligeable lors de l'implémentation de l'algorithme QRL. Comme les algorithmes Round Robin et Trade-Off se concentrent uniquement sur le délai d'exécution et ne prennent pas en compte les caractéristiques d'une tâche, la puissance résiduelle moyenne est inférieure.

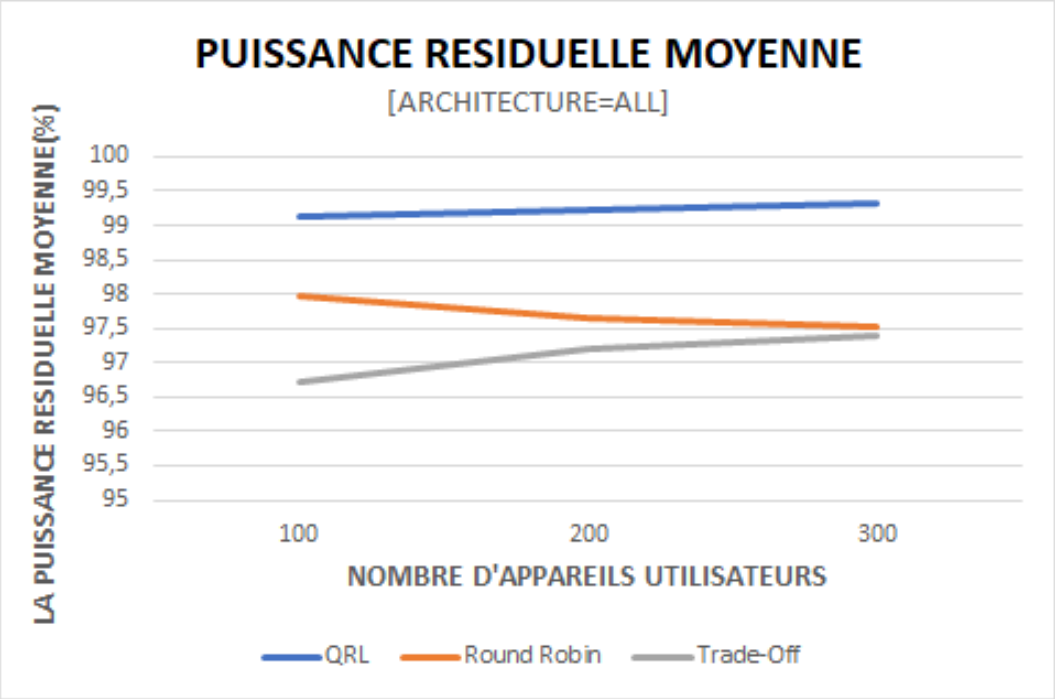

FIGURE 4.9 – La puissance résiduelle moyenne[ARCHITECTURE=ALL]

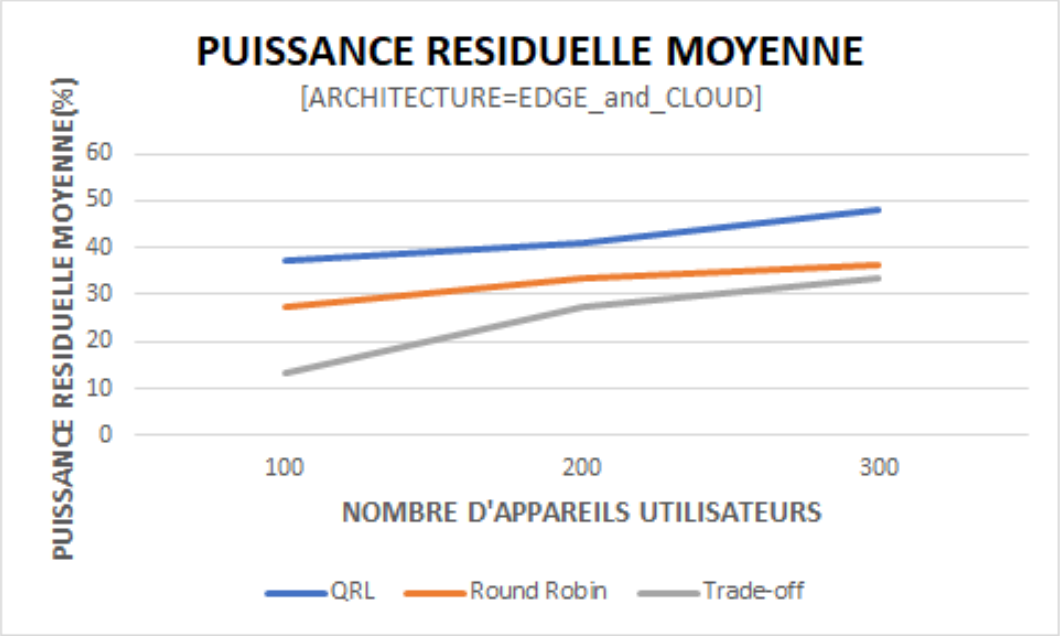

FIGURE 4.10 – La puissance résiduelle moyenne [ARCHITECTURE= EDGE_and_CLOUD]

On observe que le QRL proposé fonctionne bien dans le déchargement des tâches quelle que soit l'architecture de l'orchestrateur et malgré les contextes dynamiques impliquant des changements continus dans les charges de travail du système.

4.4 Conclusion

Ce chapitre présente la configuration des simulations considérées ainsi que les spécifications des dispositifs utilisateurs et des centres de données. Ensuite, la performance de la méthode proposée QRL en ligne est validée en comparant les résultats avec les algorithmes Round-robin et Trade-off.La faisabilité d'appliquer le framework proposé dans un environnement réaliste est validée grâce à des simulations approfondies.

Conclusion générale

Ce mémoire a examiné le problème d'allocation des ressources et du déchargement des tâches dans un environnement Edge Cloud avec l'arrivée dynamique des tâches des utilisateurs. Les paramètres de la tâche comprenaient le délai autorisé, la longueur (millions d'instructions par seconde), la taille du conteneur, la taille de la demande, la taille du résultat. Le Q Reinforcement Learning en Ligne (QRL) a été utilisé car il peut résoudre des problèmes complexes. La fonction objective comprend la consommation d'énergie et latence en tenant compte des contraintes de disponibilité des ressources.L'ordonnancement des tâches, la consommation d'énergiele, la puissance résiduelle et les échecs dû aux retards et de mobilité des appareils utilisateurs ont été analysés pour le modèle mis en œuvre.

Les résultats de la simulation ont démontré que le QRL en ligne produit de meilleures performances que les algorithmes Round Robin et Trade-Off en matière de consommation d'énergie. La disponibilité limitée des ressources et les exigences strictes de livraison en temps opportun dans diverses applications 5G nous ont motivés à utiliser la mise en cache pour améliorer les performances. La mise en cache par conteneur pour stocker les dépendances logicielles des applications populaires en fonction des coûts de stockage engagés et de la fréquence des demandes d'application a été proposée. La mise en cache à base de conteneurs a permis de réduire les défaillances et la consommation d'énergie. En outre, l'emplacement du déploiement de l'orchestrateur est également examiné, et une stratégie de déchargement distribué est proposée. Le mécanisme de déchargement décentralisé a réduit l'utilisation de la bande passante disponible, ainsi que les échecs de tâches par rapport aux autres cadres d'orchestration. En outre, l'approche décentralisée recommandée n'a pas beaucoup utilisé la puissance de la batterie de l'appareil utilisateur. Les performances exceptionnelles en termes de consommation d'énergie, de puissance de batterie restante et de défaillances de tâches ont démontré la fiabilité de l'utilisation du cache décentralisé à base de conteneurs dans un environnement réaliste.

En perspective, le DQRL et ses variants offrent de nouvelles possibilités pour l'apprentissage par renforcement en surmontant certaines limites du Q-Learning. Il permet une plus grande flexibilité et adaptabilité dans des environnements plus complexes et en constante évolution. Cependant, il reste encore des défis à relever, tels que l'exploration efficace dans de vastes espaces d'états et d'actions et la gestion des problèmes de surestimation de la fonction Q.

Bibliographie

[1] V. P. Guddeti, *Decentralized Cache-aided Offloading in Edge Cloud Collaborative Environment using Deep Q Reinforcement Learning*. PhD thesis, Carleton University, 2022.

[2] S. Chaudhary, R. Johari, R. Bhatia, K. Gupta, and A. Bhatnagar, "Craiot : concept, review and application (s) of iot," in *2019 4th international conference on internet of things : Smart innovation and usages (IoT-SIU)*, pp. 1–4, IEEE, 2019.

[3] F. Liang, W. Yu, X. Liu, D. Griffith, and N. Golmie, "Toward edge-based deep learning in industrial internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4329–4341, 2020.

[4] L. U. Khan, I. Yaqoob, N. H. Tran, S. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-computing-enabled smart cities : A comprehensive survey," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10200–10232, 2020.

[5] J. Li, Y. Ma, and Y. Wang, "Environment construction of virtual reality technology based on edge computing in immersive communication," in *2021 International Conference on Computer Technology and Media Convergence Design (CTMCD)*, pp. 117–120, IEEE, 2021.

[6] N. Hassan, K.-L. A. Yau, and C. Wu, "Edge computing in 5g : A review," *IEEE Access*, vol. 7, pp. 127276–127289, 2019.

[7] V. C. Emeakaroha, N. Cafferkey, P. Healy, and J. P. Morrison, "A cloud-based iot data gathering and processing platform," in *2015 3rd International Conference on Future Internet of Things and Cloud*, pp. 50–57, IEEE, 2015.

[8] M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra, and A. V. Vasilakos, "On optimal and fair service allocation in mobile cloud computing," *IEEE Transactions on Cloud Computing*, vol. 6, no. 3, pp. 815–828, 2015.

[9] M. B. Monir, T. Abdelkader, and E.-S. M. Ei-Horbaty, "Trust evaluation of service level agreement for service providers in mobile edge computing," in *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pp. 362–369, IEEE, 2019.

[10] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things : Architecture, advances and challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[11] N. K. Giang, R. Lea, M. Blackstock, and V. C. Leung, "Fog at the edge : Experiences building an edge computing platform," in *2018 IEEE International Conference on Edge Computing (EDGE)*, pp. 9–16, IEEE, 2018.

[12] T. K. Rodrigues, K. Suto, and N. Kato, "Edge cloud server deployment with transmission power control through machine learning for 6g internet of things," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 2099–2108, 2019.

[13] F. Jejdling *et al.*, "Ericsson mobility report," *Ericsson : Stockholm, Sweden*, 2020.

[14] C. Pallasch, S. Wein, N. Hoffmann, M. Obdenbusch, T. Buchner, J. Waltl, and C. Brecher, "Edge powered industrial control : concept for combining cloud and automation technologies," in *2018 IEEE International Conference on Edge Computing (EDGE)*, pp. 130–134, IEEE, 2018.

[15] S. Kaur and T. Sharma, "Efficient load balancing using improved central load balancing technique," in *2018 2nd international conference on inventive systems and control (ICISC)*, pp. 1–5, IEEE, 2018.

[16] K. Gai and S. Li, "Towards cloud computing : a literature review on cloud computing and its development trends," in *2012 Fourth international conference on multimedia information networking and security*, pp. 142–146, IEEE, 2012.

[17] B. P. Rimal, D. P. Van, and M. Maier, "Mobile-edge computing versus centralized cloud computing over a converged fiwi access network," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 498–513, 2017.

[18] A. H. A. Bafghi, M. Mirmohseni, F. Ashtiani, and M. Nasiri-Kenari, "Joint optimization of power consumption and transmission delay in a cache-enabled c-ran," *IEEE Wireless Communications Letters*, vol. 9, no. 8, pp. 1137–1140, 2020.

[19] S. Panwar, "Breaking the millisecond barrier : Robots and self-driving cars will need completely reengineered networks," *IEEE Spectrum*, vol. 57, no. 11, pp. 44–49, 2020.

[20] S. S. D. Ali, H. P. Zhao, and H. Kim, "Mobile edge computing : A promising paradigm for future communication systems," in *TENCON 2018-2018 IEEE Region 10 Conference*, pp. 1183–1187, IEEE, 2018.

[21] B. Panchali, "Edge computing-background and overview," in *2018 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 580–582, IEEE, 2018.

[22] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[23] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[24] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks : A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[25] J. Zhang, X. Zhou, T. Ge, X. Wang, and T. Hwang, "Joint task scheduling and containerizing for efficient edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 2086–2100, 2021.

[26] H. Wan, "Deep learning : neural network, optimizing method and libraries review," in *2019 International Conference on Robots & Intelligent System (ICRIS)*, pp. 497–500, IEEE, 2019.

[27] N. Praveena and K. Vivekanandan, "A review on deep neural network design and their applications," in *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1, pp. 1495–1501, IEEE, 2021.

[28] W. Xu, L. Chen, and H. Yang, "A comprehensive discussion on deep reinforcement learning," in *2021 International Conference on Communications, Information System and Computer Engineering (CISCE)*, pp. 697–702, IEEE, 2021.

[29] W. T. Scherer, S. Adams, and P. A. Beling, "On the practical art of state definitions for markov decision process construction," *IEEE Access*, vol. 6, pp. 21115–21128, 2018.

[30] M. Naeem, S. T. H. Rizvi, and A. Coronato, "A gentle introduction to reinforcement learning and its application in different fields," *IEEE access*, vol. 8, pp. 209320–209344, 2020.

[31] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa," *IEEE Access*, vol. 8, pp. 54074–54084, 2020.

[32] S. Nath and J. Wu, "Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems," *Intelligent and Converged Networks*, vol. 1, no. 2, pp. 181–198, 2020.

[33] J. Yao, T. Han, and N. Ansari, "On mobile edge caching," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2525–2553, 2019.

[34] M. Othman, S. A. Madani, S. U. Khan, *et al.*, "A survey of mobile cloud computing application models," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 393–413, 2013.

[35] M. Y. Garg and M. B. Gupta, "Performance analysis and comparison of the micro virtual machines provided by the top cloud vendors.,"

[36] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Edge intelligence for energy-efficient computation offloading and resource allocation in 5g beyond," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 12175–12186, 2020.

[37] O.-K. Shahryari, H. Pedram, V. Khajehvand, and M. D. TakhtFooladi, "Energy and task completion time trade-off for task offloading in fog-enabled iot networks," *Pervasive and Mobile Computing*, vol. 74, p. 101395, 2021.

[38] T. K. Rodrigues, K. Suto, and N. Kato, "Edge cloud server deployment with transmission power control through machine learning for 6g internet of things," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 2099–2108, 2019.

[39] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, pp. 50–56, 2016.

[40] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. Inácio, and M. M. Freire, "Cloudsim plus : a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness," in *2017 IFIP/IEEE symposium on integrated network and service management (IM)*, pp. 400–406, IEEE, 2017.

[41] J. Ikuno, M. Wrulich, and M. Rupp, "3gpp tr 36.814 v9. 0.0-evolved universal terrestrial radio access (e-utra) ; further advancements for e-utra physical layer aspects," *Tech. Rep.*, pp. 90–103, 2010.