TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ALGORITHMES

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

IoT	Internet Of Things
IdO	Internet des Objets
RAM	Random Access Memory
CPU	Central Processing Unit
M2M	Machine to Machine
WSN	Wireless Sensor Network
MCC	Mobile Cloud Computing
CAP	Central Access Point
DT-IoTCM	Distribution of Tasks among IoT nodes, Containers and Mobile phones
DT-IoTM	Distribution of Tasks between IoT nodes and Mobile phones

LISTE DES SYMBOLES ET UNITÉS DE MESURE

W	Watt
MHz	MegaHertz
MQTT	Message queuing telemetry transport
GHz	GigaHertz
KB	KiloByte
GB	GigaByte
Mbps	Megabit par seconde
J	Joule
s	Seconde

CHAPITRE 1

INTRODUCTION

1.1 Contexte et motivation

La domotique et la maison intelligente visent à offrir une vie plus sure et aisée en automatisant les foyers avec le minimum d'interventions humaines. Une maison intelligente est généralement équipée de caméra et des capteurs pour surveiller les conditions domestiques telles que la température, l'humidité, le bruit, la lumière, la qualité d'air, etc, tout en respectant les normes de sécurité et de confort, et améliorant la qualité de vie. Des nombreuses entreprises se précipitent pour devenir les pionnières dans le domaine IdO en offrant aux utilisateurs des services de contrôle de leurs maisons intelligentes accessibles à travers une application mobile. Vu la demande exponentielle simultanée, et en temps réel de ces services, ces entreprises ont recours à l'informatique en nuage mobile afin de centraliser ces services et les rendre accessibles n'importe où et n'importe quand (Stojkoska et al., 2017).

Afin d'offrir un service de control d'une maison, trois plateformes doivent être mises en œuvre. (Figure.1-1):
Une plateforme des périphériques IdO, une plateforme pour la collecte de données de contrôle et une plateforme des appareils mobiles récepteurs de ces données en temps réel. Le bon fonctionnement de ces plateformes est une nécessité primordiale pour contrôler la maison. En effet, chaque plateforme a des contraintes spécifiques. Par exemple des nœuds IdO comme Raspberry pi, Arduino, ESP8266, etc, (Barbon et al., 2016) sont limités en ressources (RAM et CPU) et en batteries. Le problème s'aggrave quand le nombre des capteurs attachés, les tâches à traiter ainsi que les données à fournir augmentent. Le même problème de limitation de ressources et de batterie est présent pour les appareils mobiles. Vu la capacité bornée de CPU de ces composants, le traitement abondant des tâches engendre une grande consommation énergétique et une utilisation débordée des ressources. C'est pour cela la surcharge des

ressources peut être considérée comme un facteur de performance essentiel. Du côté infonuagique, la centralisation de ces services n'est pas assez bénéfique puisqu'elle peut être coûteuse quand l'utilisation augmente.

L'envoi des données relatives à l'exécution des tâches sur l'infonuagique ou l'appareil mobile est soumis à la contrainte de capacité réseau. Puisque les données sont initialement dans les nœuds IdO, leur déchargement vers une autre plateforme distante dépend de l'état actuel du réseau. Tel que c'est prouvé, le déchargement de données consomme une quantité d'énergie assez importante (Chen, 2014).

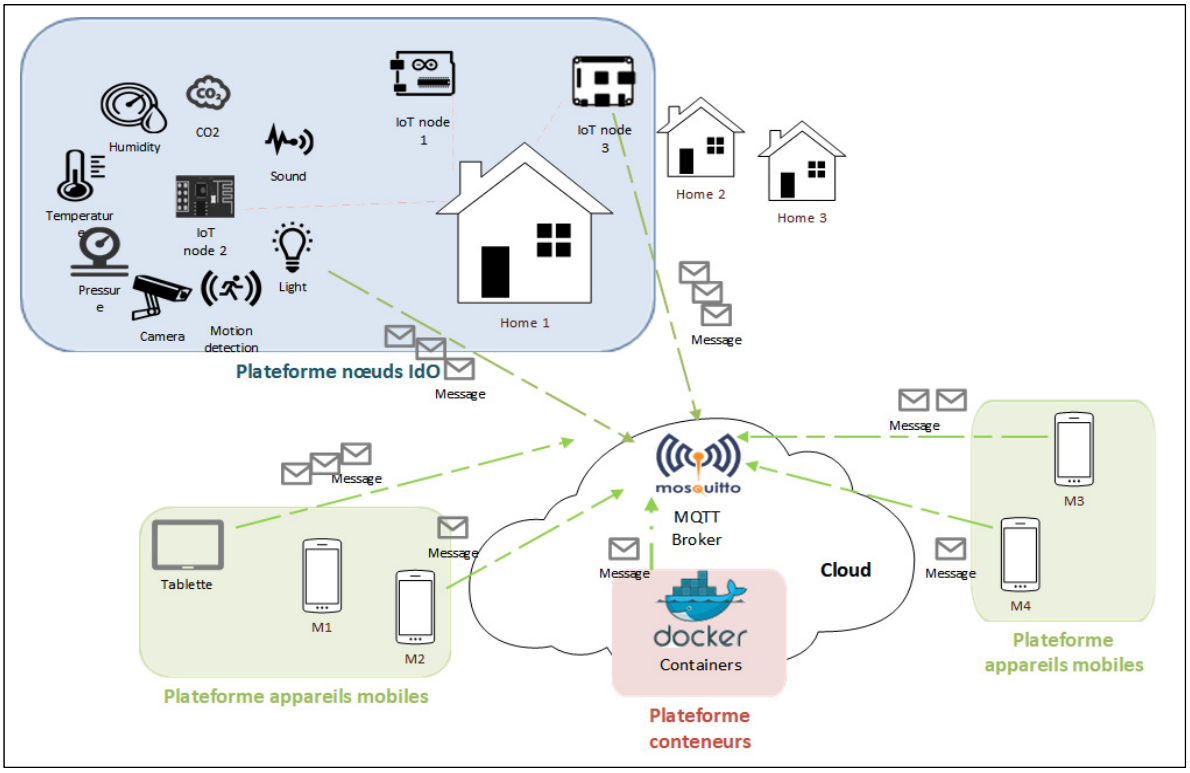

Figure 1-1 Vue globale d'une application IdO pour la maison intelligente

De nos jours, les architectures adaptives constituent un élément clé de toute application à grande échelle connue. La complexité de la gestion de telles applications est exacerbée lorsque l'application est exécutée sur différentes plateformes y compris l'infonuagique. D'où, le

développement d'une application en tant que suite de petits services est devenu nécessaire (Martin Fowler, 2016). Les microservices représentent une solution appropriée qui permet à une organisation de répondre aux exigences des applications IdO tout en permettant le développement et le déploiement plus rapides et plus simples.

Dans notre système, on a suivi cette architecture pour implémenter nos microservices. Les **microservices** sont basés sur des modules**.** Ils consistent un modèle de "conception" qui guide la mise en œuvre de la fonctionnalité. Ils se servent sur des files d'attente, des descripteurs de fichiers (tubes, sockets UNIX, etc.), sur des messages, sur le réseau et même des courriers électroniques.

Pour la plateforme des nœuds IdO, on a adapté la notion des microservices en respectant l'aspect modulaire des tâches et en assurant une faible indépendance entre les microservices. Prenant l'exemple d'un capteur de la température qui fournit un service appelé 'control de la température'. Ce service est décomposé en un ensemble des tâches considérées microservices. Ces tâches sont : temperature_raw-reader, temperature_preprocessing, temperature_collection, temperature_analysis, temperature_notification et temperature_visualization. Inspirée par l'application Android d'Amazon (Dzone, 2016) et (ProAndroidDev,2019), la plateforme des appareils mobiles suit aussi l'architecture des microservices.

Le problème se focalise maintenant sur les microservices et les tâches de l'application.

Vu leur capacité de traitement limitée, une affectation des tâches équitable sur les microservices doit être fournie. Elle permet de minimiser la consommation énergétique totale et assurer la qualité de service souhaitée des applications toute en optimisant l'utilisation des ressources de nœud IdO, l'infrastructure infonuagique et l'appareil mobile. Sans oublier l'adaptation au changement de la bande passante et le respect de la durée de traitement limité d'une tâche.

1.2 Problématique

Prenant l'exemple d'une maison intelligente équipée par des nœuds IdO installés dans chaque chambre qui détectent les données relatives à la température, humidité, CO2, lumière…

Le suivi de la maison se fait via une application mobile installée chez les utilisateurs. Les différentes tâches formant l'application possèdent leurs correspondants en microservices. Le choix de l'architecture microservices revient à son adaptation avec la technologie d'internet des objets, son aspect évolutif et l'hétérogénéité d'un tel milieu distribué où chaque plateforme utilise un langage de programmation diffèrent. L'application IdO est décomposée en des petits services indépendants, autonomes et modulaires. Chaque module prend en charge un objectif métier spécifique et utilise une interface simple pour communiquer avec d'autres plateformes et modules. Autrement dit, chaque microservice représente une tâche bien précise. Comme les tâches suivantes : **lecteur de données brutes** (data_raw_reader) qui lit les données de différents capteurs (température, humidité, CO2, lumière, etc.), **le prétraitement des données** (data_preprocessing) convertit les données en données compréhensibles, **la collecte de données** (data_collection) collecte les données et les enregistre sur la table de base de données appropriée (table de température, table d'humidité, etc.), **analyse de données** (data_analysis) qui fait la comparaison avec des valeurs seuils afin de détecter tout comportement étrange d'un capteur , **notification de données** (data_notification) sert à notifier l'utilisateur, et **visualisation des données** (data_visualization) est un client abonné MQTT qui demande un accès aux données du domicile afin de visualiser leurs graphes.
Le tableau 1-1 récapitule l'ensemble des tâches (microservices) et leurs plateformes de traitement possibles.

Tableau 1-1 Les microservices d'une application IdO et ses possibles emplacements

Microservices	Nœud IoT	Appareil mobile	Infonuagique
Data_raw_reader	X		
Data_preprocessing	X	X	X
Data_collection		X	
Data_analysis	X	X	X
Data_notification	X	X	X
Data_visualization		X	

Ces microservices sont implémentés sur les trois plateformes de système : La plateforme équipée des nœuds IdO et ses microservices sont développés sous Arduino.

La plateforme infonuagique menée des microservices lancés dans des conteneurs Docker. Et la plateforme des appareils mobiles où ses microservices sont développés sous Android. Ils communiquent tous de façon asynchrone en utilisant le protocole MQTT. Tandis que l'application mobile on l'a développé de façon à utiliser les API REST pour une communication synchrone et le MQTT pour la communication asynchrone (figure 1-2).

Une tâche peut activer et désactiver un microservice selon le besoin.

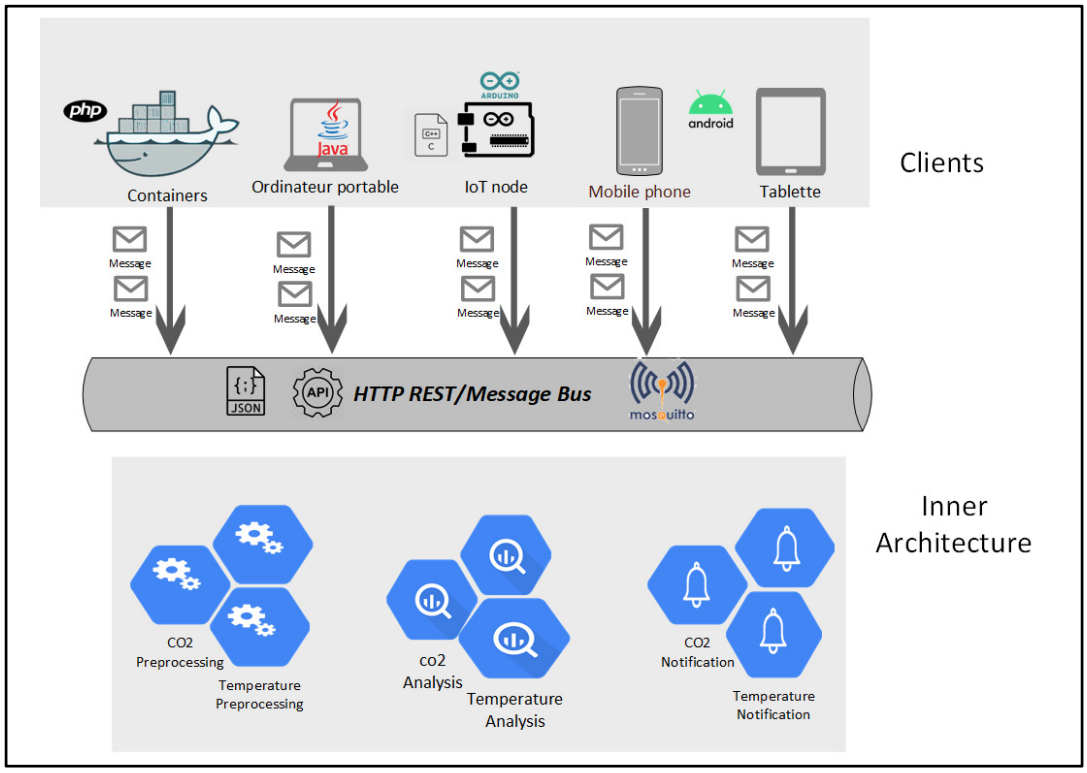

Figure 1-2 Architecture microservices adaptée pour une application IdO

Dans ce contexte, l'exécution des tâches relatives à cette application peut être partagée entre les trois plateformes. Malheureusement, le nombre des tâches peut augmenter. Chaque équipement participant est doté des capacités de batteries et de ressources différentes. Un traitement excessif des tâches peut saturer les équipements et engendrer une consommation énergétique inutile. De plus, les données échangées entre deux équipements communicants sont

affectées par l'état actuel du réseau (e.g., bande passante, délai). Alors, une mauvaise gestion peut dégrader la performance d'une telle application IdO.

Dans ce travail, nous proposons une solution optimale pour le problème de distribution des tâches sur les plateformes par DT_IoTMC (Distribution of tasks among IoT nodes, Mobile phones and Cloud containers). L'objectif principal de DT_IoTMC est de minimiser la consommation énergétique globale du système en distribuant les tâches aux équipements appropriés sans surcharger ses ressources et drainer ses batteries tout en s'adaptant à l'état actuel du réseau et respectant les exigences des applications IdO et la qualité de service demandée. Cette solution va être implémentée sur l'infonuagique dont les paramètres d'entrées sont les suivants : le nombre des tâches, l'utilisation en ressources de chaque tâche, la taille des données échangées d'une tâche, les capacités des ressources et des batteries de chaque équipement du système ainsi que le délai d'échéance de chaque tâche. Le résultat sera un plan d'exécution optimal des tâches sur les microservices disponibles et la consommation minimale d'énergie.

1.3 Question de recherche

Afin de bien cerner notre travail, nous posons les questions de recherche suivantes :

- ✓ **RQ1 :** Comment peut-on implémenter les microservices dans les plateformes de contrôle de maison intelligente afin de bien manipuler les tâches et les données échangées?

  La nouvelle solution doit assurer un faible couplage et une grande autonomie entre les charges de travaux décentralisées dans les trois plateformes.

- ✓ **RQ2 :** Comment doit-on choisir le lieu d'exécution et la distribution des tâches dans un tel milieu hétérogène composé de trois plateformes (nœuds IdO, conteneurs sur le nuage et appareils mobiles) :
  - o En diminuant la consommation totale d'énergie,

- En respectant les exigences des applications de contrôle des maisons intelligentes,
- En assurant la qualité de service et l'expérience souhaitée,
- Sans surcharger et drainer les ressources (RAM et CPU) de chaque équipement?

- ✓ **RQ3 :** est-ce que la solution de contrôle peut s'adapter au changement de l'état de notre système afin de toujours maintenir une distribution équitable des tâches et une énergie minimale consommée?

1.4 Objectifs

L'objectif principal de notre recherche est de concevoir un système qui optimise la distribution des tâches de contrôle de maison intelligente sur trois plateformes : plateforme des IdO, des appareils mobiles et d'infonuagique tout en minimisant la consommation totale de l'énergie, en tenant compte de l'état actuel de système (état du réseau) et en respectant les exigences des services à fournir, sans drainer et surcharger les ressources des équipements impliqués (les nœuds IdO, l'infrastructure en nuage et les appareils mobiles).
Cet objectif peut se traduire en trois sous objectifs, comme suit :

- ✓ **O1** : La modélisation de la distribution des tâches de contrôle de la maison intelligente sur les microservices exécutés sur trois plateformes.

- ✓ **O2** : La formulation mathématique d'un problème d'optimisation qui minimise l'énergie consommée lors de l'exécution de tâches tout en s'adaptant à l'état actuel du réseau, et en respectant les capacités de ressources des équipements sans drainer les batteries et sans violer les échéances des tâches.

- ✓ **O3** : La mise en place d'une solution pour suivre la variation du système, et distribuer les tâches d'une façon optimale et convenable sur les équipements de différentes plateformes.

1.5 Contributions

Dans ce contexte, les principales contributions de cette recherche sont résumées comme suit:

- Une formulation de problème de distribution des tâches est présentée. Cette dernière vise à minimiser la consommation totale d'énergie d'un système composé de trois plateformes équipées par des équipements multiples en tenant compte de l'état actuel du réseau, de la capacité des ressources des équipements, des batteries et des exigences des tâches de l'application IdO.
- Deux algorithmes pour résoudre ce problème :
    - Un algorithme pour le déchargement hybride qui prend la décision de traiter les tâches localement sur les nœuds IdO ou de les décharger vers les appareils mobiles ou vers les conteneurs de l'infonuagique.
    - Un algorithme pour le déchargement deux tiers qui autorise le traitement local ou le déchargement vers les appareils mobiles.
- Une application IdO qui suit l'architecture microservices est proposée. Elle est composée d'une plateforme équipée des nœuds IdO, d'une plateforme équipée des appareils mobiles et une plateforme infonuagique composée des conteneurs. Deux modules sont conçus et implémentés. Un module de collecte des profileurs de différents équipements système. Un module d'optimisation qui prend et applique une décision optimale pour chaque changement critique de l'état du système.

1.6 Plan du mémoire

Notre mémoire est divisée en quatre chapitres selon la structure suivante :

- Le premier chapitre est une introduction générale. Nous présentons d'abord le contexte général et les motivations de cette recherche. Ensuite, l'énoncé du problème, les défis associés et puis les objectifs à atteindre.

- Le deuxième chapitre est intitulé l'état de l'art. Il présente les différentes technologies indispensables pour bien cerner notre projet de recherche. Les différentes architectures adoptées pour présenter une application IdO. Ainsi que les travaux connexes qui ont abordé un problème analogue à notre problème comme le déchargement des tâches vers une entité distante et leurs impacts sur la consommation d'énergie et les ressources utilisées dans un tel domaine de l'internet des objets et le système distribué. Une comparaison des approches existantes et ses limites a été réalisée afin de mettre en évidence les contributions de ce mémoire.
- Le troisième chapitre est consacré à la méthodologie. Selon les objectifs de notre thèse, la première partie est dédiée à proposer une nouvelle architecture pour le système étudié et de le modéliser. La seconde partie traite le modèle d'optimisation offert. La troisième partie présente les algorithmes proposés. Finalement, le déploiement de solutions proposées.
- Le quatrième chapitre présente d'abord la mise en œuvre du système proposé, puis examine les configurations expérimentales et les résultats de la simulation.

CHAPITRE 2

ÉTAT DE L'ART

2.1 Introduction

Dans ce chapitre, nous présentons le contexte technique de ce mémoire telles que les architectures utilisées dans des applications IdO, précisément l'architecture microservice et les différents travaux de recherche connexes liés aux problèmes de prise de décision de déchargement, de distribution des tâches, de consommation d'énergie et des ressources utilisées. En conséquence, nous discutons leurs principaux avantages et inconvénients qui donneront naissance à notre approche proposée.

2.2 Architectures des applications IdO

L'architecture des applications IdO et mobile suit l'architecture logicielle. Elle est décrite comme un processus évolutif. Cette évolution est due aux limitations et/ou failles rencontrées. Les itérations successives du développement de logiciels aboutissent à de meilleures technologies et de meilleures approches. D'où, l'apparition d'une nouvelle architecture est une nécessité inévitable qui répond mieux aux attentes des utilisateurs.

2.2.1 Architecture monolithique

L'architecture monolithique représente l'ancêtre de toutes les architectures logicielles. Avec ce type d'architecture, l'application proposée était regroupée dans un seul gros fichier, dans laquelle différents composants, plusieurs services sont combinés en un seul programme indissociable à partir d'une seule plateforme.

Les applications d'entreprise sont construites de trois parties : une base de données constituée de nombreuses tables situées dans un système de gestion de base de données relationnelle, une interface utilisateur cotée client et une interface cotée serveur.
Une filiale de 'Samsung Electronics' appelée 'SmartThings' propose un grand réseau de périphériques IoT spécialisé dans les maisons intelligentes. Il donne aux clients une gamme de capteurs, d'applications numériques (sur les appareils mobiles) et d'appareils intelligents capable de contrôler, de surveiller et d'automatiser les systèmes de sécurité, les prises de courant, les électroménagers, les éclairages… La solution proposée repose sur l'architecture monolithique qui a rencontré des problèmes à suivre le rythme de la demande croissante des utilisateurs (Objectcomputing, 2019).

2.2.2 Architecture orientée service (SOA)

Les problèmes rencontrés avec l'architecture monolithique ainsi que les problématiques d'interopérabilité concernant les technologies informatiques utilisées en entreprise ont posé des défis. Ce qui a donné naissance à ce terme architecture orientée service (AOS ou SOA). Cette architecture est considérée comme un modèle d'interaction applicative distribué qui expose des composants logiciels (services). Elle suit le principe de fournisseur et consommateur de service. L'adoption des principes SOA permet de décomposer des systèmes complexes et monolithiques en des applications composées d'un écosystème de composants plus simples et bien définis. L'utilisation d'interfaces communes et de protocoles standard donne une vue horizontale d'un système d'entreprise (Atzori et al., 2010).
Cette architecture utilise des formats d'échange (XML ou JSON) et une couche d'interface interopérable connue sous le nom de service web.
Les avantages de l'approche SOA sont reconnus dans la plupart des études par les solutions d'intergicielles pour l'IdO. TinySOA est un exemple d'intergiciel pour l'architecture orientée service. Il permet aux programmeurs d'accéder au WSN à partir de leur application, à l'aide d'une simple API orientée service.

L'objectif général de TinySOA est de faciliter l'accès au réseau de capteurs sans fil et de les incorporer dans la mise en œuvre de l'application (Bin Abd Rahim *et al*., 2018).

2.2.3 Architecture microservices

L'année 2014 est considérée comme une année révolutionnaire pour les entreprises et les architectures logicielles qu'elles utilisent. Le développement d'applications mobiles, Big data, infonuagique, API et les objets connectés a posé des problèmes d'intégration (Lemagit, 2014).

En contrepartie, le développement de systèmes monolithiques robustes a atteint ses limites, car la mise en œuvre des changements dans les systèmes actuels de grande taille, complexes et à évolution rapide serait trop lente et inefficace. En réponse à ces problèmes, l'architecture de microservice a émergé et elle est rapidement devenue une solution largement utilisée. Une telle architecture modulaire convient aux environnements distribués qui utilisent des solutions Internet des objets (Krivic *et al*., 2018).

Le terme microservice décrit un ensemble de petits services autonomes, chacun réalise son processus métier de manière autonome et communique de façon synchrone via un API ou asynchrone via un bus de messagerie léger (MQTT).

La différence entre une telle architecture et l'approche monolithique classique se localise dans le fait qu'une application est décomposée en des fonctions clés isolées appelées microservices. Ces microservices peuvent être déployés indépendamment dans des machines de déploiement entièrement automatisées autour des capacités de l'entreprise (Microsoft, 2018). Ils peuvent utiliser différentes technologies de stockage de données et être écrits dans différents langages de programmation. Son point fort défini par Sam Newman dans son livre « Building Microservices. » (Newman, 2014), est que les microservices sont « petits, se concentrent pour faire convenablement une seule tâche » (Lewis et Fowler, 2014).

En réalité, l'architecture des microservices ressemble beaucoup à l'architecture orientée services. Une interface REST / SOAP peut être fournir aux clients. Mais en interne, ce point de terminaison REST est implémenté sous la forme d'un microservice.

Cependant, grâce aux technologies de conteneurisation, les microservices sont devenus plus viables. Cette technologie donne la possibilité de lancer et exécuter indépendamment différentes parties d'une application sur la même machine mais avec un niveau de contrôle bien plus élevé sur leurs éléments et cycles de vie. Avec les API et les pratiques DevOps, les microservices conteneurisés constituent la base des applications natives pour le cloud (RedHat, 2017).

Parmi les avantages de cette architecture, on cite (Médini,2015):

- L'évolutivité verticale en permettant les réplications seulement des services chargés et l'évolutivité horizontale qui permet la migration des services les plus chargés vers des nœuds différents.
- La possibilité d'utiliser plusieurs socles techniques par service (par exemple une base de données NoSql avec du NodeJs d'un côté, API web ASP.NET core de l'autre ...)
- Le déploiement indépendant (mise à jour d'un service sans déployer tout le reste).
- Agilité et isolation des fonctionnalités
- Possibilité d'utiliser des versions différentes de dépendances dans des services différents (bibliothèques, version de produits, etc.)

Malgré ces avantages évidents, l'architecture de microservices emmène un nouvel ensemble de problèmes, particulièrement en ce qui concerne les tests, la surveillance et le débogage d'une application nouvellement décentralisée et faiblement couplée. Si les services sont trop petits ou trop grands, assurer un équilibre parfait sera difficile pour les développeurs. D'où, un bon type d'automatisation et d'outils peut résoudre tous les inconvénients ci-dessous.

Plusieurs compagnies ont parlé de leurs expériences en suivant cette architecture comme Netflix, Uber, EBay, Amazon, Sound Cloud, … Une plateforme IoT smart city a été créée en appliquant l'architecture de microservices pour une variété d'applications afin d'accroitre l'efficacité énergétique d'une ville (Krylovskiy et al., 2015). Une architecture basée sur les microservices et axée sur la connexion a été proposée afin de résoudre les problèmes d'interopérabilité et d'évolutivité (Vresk & Cavrak, 2016). Elle est basée sur l'orchestration

de différents composants hétérogène du système IoT (périphériques, sources de données, processeurs de données, stockage, etc.).

La motivation d'adopter une telle architecture dans ce travail est tirée de la présence d'une grande similitude entre l'architecture des microservices et les systèmes IdO.
Le tableau 2-1 résume les caractéristiques des trois différentes architectures (dzone, 2017), (capgemini, 2016).

Tableau 2-1 Tableau comparatif des trois architectures logicielles

Architecture monolithique	**Architecture orientée service**	**Architecture microservices**
Applications étroitement couplées	Composants plus faiblement couplés qui peuvent être utilisés dans différents contextes.	Des services applicatifs indépendants offrant une seule capacité métier de manière indépendante, faiblement connectée et autonome.
Toutes les modifications doivent être poussées à la fois	Permet d'apporter des modifications à des pièces individuelles. Mais chaque pièce doit être soigneusement modifiée pour s'intégrer à la conception globale.	La création, la maintenance et l'amélioration de nouveaux services sont faites de manière indépendante.
Un déploiement continu est presque impossible	Exposer des composants discrets d'une application en tant que services Web	Liaison d'informations via API de données partagée.
Une seule unité	Granularité grossière	Granularité fine

2.3 La décision de déchargement des flux et ses impacts

D'une part, les applications IdO deviennent plus complexes chaque fois que le nombre des capteurs et le nombre des demandeurs des services augmentent. Les données fournies par un capteur de CO2 sont des données critiques et qui doivent être transmises et traitées en temps réel. Les services offerts peuvent être gourmands de côté ressources (RAM et CPU) et énergie.

D'autre part, il est connu que les périphériques d'extrémité (comme les capteurs, les équipements IdO et les appareils mobiles) ont des ressources limitées, telles que la durée de vie de la batterie, la bande passante du réseau, la capacité de stockage et les performances du processeur (Kumar *et al.,* 2013). Ces exigences ont donné une nécessité pour les futures recherches pour qu'elles prennent en considération le lancement d'un service dans un équipement sans excéder ses capacités et sans dissiper une grande quantité d'énergie. Ainsi que le déchargement de sa charge de travail dans un autre environnement distant doit respecter ses contraintes.

L'approche de déchargement est survenue pour alléger ces restrictions. Les universités et l'industrie ont pris en compte cette technologie prometteuse afin de libérer l'utilisation des ressources des appareils telles que la puissance de calcul, l'énergie, le stockage… Brièvement, le déchargement permet d'envoyer ou transférer des tâches de calculs lourds à des équipements ou plateformes distants dotés de ressources abondantes.

2.3.1 La prise de décision et les types de déchargement

Au cours de la dernière décennie, plusieurs types de déchargement ont été proposés selon les problèmes rencontrés. La plupart des déchargements ont considéré un appareil mobile comme l'équipement faible en ressources, initiatif de processus de déchargement.

2.3.1.1 Déchargement deux tiers

Cette approche consiste à donner la possibilité à un appareil mobile de décharger une partie de sa charge de travail sur un serveur infonuagique via un ou plusieurs réseaux de communication afin de tirer parti des ressources excessives de l'infonuagique.
Ce type de système de déchargement dépend essentiellement de la fiabilité des communications de bout en bout et de la disponibilité de l'infonuagique (Shu et al., 2013). L'accès au nuage est souvent affecté par des facteurs incontrôlables, tels que l'instabilité et l'intermittence des réseaux sans fil ainsi qu'un temps de latence élevé.

2.3.1.2 Déchargement trois tiers

Le déchargement à deux niveaux devient inapproprié dans le cas où les temps de latence deviennent critiques. Des recherches ont proposé un nouveau déchargement appelé déchargement trois tiers. Afin de remédier à la pénurie de ressources d'un appareil mobile (par exemple la batterie) et l'exigence d'une certaine charge de travail considérée sensible au temps, des chercheurs ont eu recours à un équipement intermédiaire à proximité de l'appareil mobile et riche en ressources (par exemple nuage local, MEC ou cloudlet).
Ce type de déchargement réalise en premier lieu un déchargement sur l'intergiciel (middlware) en utilisant le WiFi ou une connexion radio à courte portée, puis une migration sur l'infonuagique distante.
Le Mobile Edge Computing (MEC) est venu pour remédier au problème de délai de déchargement des données puisque le nuage est vu comme un équipement loin des utilisateurs mobiles (Chen & Zhang., 2017). Toutefois, en cas d'un système dense, les MEC sont aussi limités par leurs capacités de calcul et les ressources par rapport aux MCC. Par conséquent, des chercheurs pensent que la prise de décision sur le lieu de déchargement peut faire une différence et amener un plus si à la fois on prend en considération les avantages de nuage et de Mobile Edge.

2.3.1.3 Déchargement hybride

Ce type de déchargement est nouveau et rarement discuté dans les anciens travaux de recherche. Afin d'augmenter le spectre des moments opportunistes pour décharger une tâche, des nouvelles stratégies sont mises en place (Flores *et al.,* 2017), comme le système hybride schématisé dans la figure 2-1 qui intègre et unifie tous les types de systèmes de déchargement. L'objectif est d'accroître la disponibilité de la prise en charge du déchargement pour un appareil mobile en fusionnant les déchargements cloudlet, de périphérique à périphérique et à distance. Dans ce contexte, un cadre hybride appelé HyFog est proposé. Ce dernier donne la possibilité aux utilisateurs de choisir de façon flexible l'exécution locale sur l'appareil mobile, le déchargement D2D (devise to devise) et l'exécution et le déchargement vers l'infonuagique. Dans ce fait, un algorithme de graphes à trois couches pour un déchargement efficace des tâches hybrides entre les périphériques est mis en place afin de minimiser le coût total d'exécution des tâches.

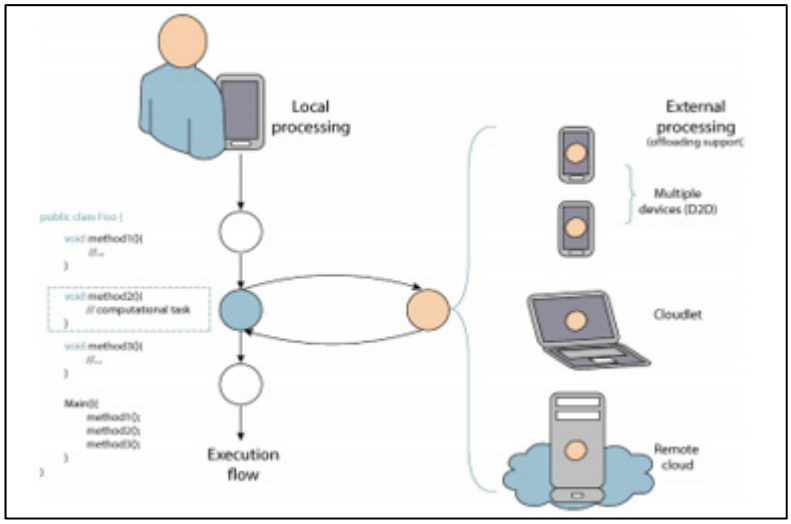

Figure 2-1 Système de déchargement informatique hybride
((Flores *et al.,* 2017)

Un cadre informatique CAME (Ma *et al.,* 2017) est proposé dans la figure 2-2, dans lequel des ressources de l'infonuagique étaient louées pour améliorer la capacité de calcul du système, tandis que des ressources Edge mobiles étaient utilisées pour réduire le temps de la-

tence. Le système CAME a modélisé le retard sous la forme d'un réseau de file d'attente, et les décisions de déchargement vont être prises en optimisant à la fois l'utilisation des ressources du nuage et équilibrant les charges de travail entre le bord mobile et le nuage.

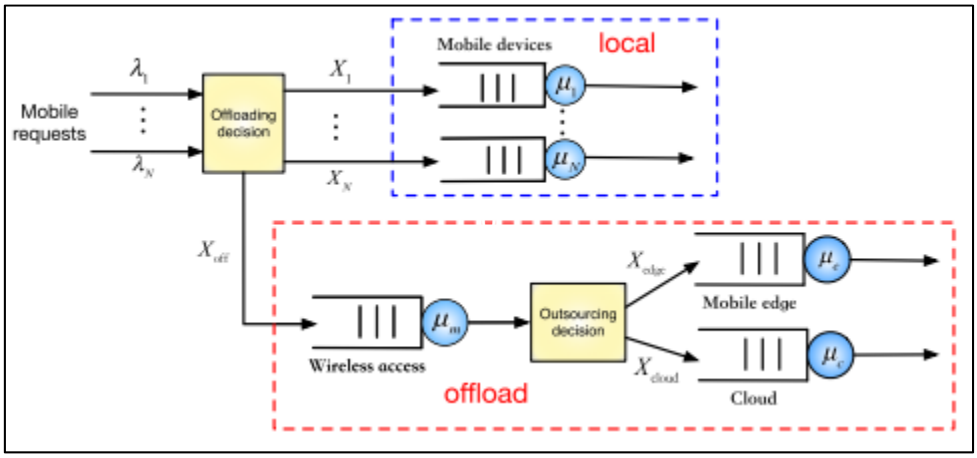

Figure 2-2 Un système hybride de déchargement (Mobile à bord ou nuage) (Ma et al., 2017) et (WU, 2018)

2.3.2 Impact de déchargement sur la consommation d'énergie

Beaucoup de travaux antérieurs ont eu recours à l'approche de déchargement distant afin de minimiser la consommation énergétique des équipements du système.

L'application mobile a été considérée comme une séquence de tâches qui peuvent être exécutées en collaboration entre le périphérique et l'infonuagique clone (Zhang *et al.,* 2015). L'objectif est la minimisation de la consommation énergétique sur le périphérique mobile et le déchargement de manière stratégique des tâches sur l'infonuagique tout en respectant leurs délais de traitement. Le problème est considéré comme un problème de chemin le plus court. La conclusion que Zhang et al ont tiré que l'exécution en collaboration des tâches permet de prolonger la durée de vie de batterie des périphériques en réduisant considérablement leurs consommations énergétiques. La limitation de ce travail se résume dans le fait que le coût de communication n'est pris en considération.

L'application mobile peut être considérée comme étant un graphe de coût divisé en des tâches serveur et client (Zhang & Wen ,2018). Un algorithme Branch and Bound est proposé

pour résoudre le problème d'optimisation et minimiser la consommation d'énergie du calcul et le coût de communication des données pour l'exécution de tâches. Le coût énergétique est formulé en fonction des paramètres suivants : l'état du réseau, l'état de CPU et l'état d'affichage afin de prendre la décision pour migrer ou décharger les tâches de l'application (Chun *et al.,* 2011). Le problème d'optimisation est formulé sous la forme d'un programme linéaire entier qui a donné de bons résultats du côté énergie sans garantir le temps d'exécution.

Chen et al ont considéré un système avec multi-utilisateurs où chaque utilisateur a plusieurs tâches indépendantes. Le problème d'optimisation à objectifs multiples a été simplifié en tant que problème d'optimisation à objectif unique avec des poids différents pour différents objectifs afin de minimiser le coût total de l'énergie du système. (Chen et al., 2018).

Dans ce contexte, ils ont proposé un algorithme efficace en trois étapes (déchargement trois tiers) afin de décider le lieu d'exécution de chaque tâche. C.-à-d.., l'exécution locale sur l'appareil mobile, l'exécution sur le point d'accès partagé CAP ou l'exécution sur l'infonuagique.

$$\min_{\{\mathbf{x}_{ij}\}} \quad \sum_{i=1}^{N} \left[\sum_{j=1}^{M} (E_{ij}^{l} x_{ij}^{l} + E_{ij}^{A} x_{ij}^{a} + E_{ij}^{C} x_{ij}^{c}) + \rho_i \max\{T_i^{L}, T_i^{A_{(U)}}, T_i^{C_{(U)}}\} \right] \tag{2.1}$$

N et M représente respectivement le nombre des utilisateurs et des tâches. La variable décisionnelle $x_{i,j}= \{x_{i,j}^{l}, x_{i,j}^{a}, x_{i,j}^{c}\}$ qui dicte si la tâche i reliée à l'appareil mobile j est exécutée localement, sur le CAP partagé ou sur le nuage. $E_{i,j}^{L}, E_{i,j}^{A}$ et $E_{i,j}^{C}$ expriment la quantité d'énergie consommée lors de l'exécution locale, sur le CAP ou sur l'infonuagique de la tâche i reliée à l'appareil mobile j. ρ_i représente le poids sur le délai de traitement des tâches par rapport à la consommation d'énergie dans le coût total du système.

T_i^{L} représente le délai de traitement des tâches traitées par l'utilisateur mobile i lui-même. $T_i^{A(U)}$ et $T_i^{C(U)}$ représentent respectivement la somme directe des délais de transmission et des délais de traitement sans chevauchement. Ils sont toujours supérieurs au retard réel étant donné la même décision de déchargement et la même allocation de ressources.

L'équation objective est soumise à ces contraintes :

$$\sum_{i=1}^{N} c_i^u \leq C_{\mathrm{UL}}, \tag{2.2}$$

$$\sum_{i=1}^{N} c_i^d \leq C_{\mathrm{DL}}. \tag{2.3}$$

Ces contraintes exigent que la bande passante de liaison montante c_i^u et de liaison descendante c_i^d allouée à l'utilisateur i sont limitées par la bande passante C_{UL} de la liaison montante et la bande passante C_{DL} de la liaison descendante.

$$\sum_{i=1}^{N} (c_i^u + c_i^d) \leq C_{\mathrm{Total}}. \tag{2.4}$$

Cette contrainte exige que la somme de la bande passante de liaison montante et descendante ne dépasse pas la capacité totale de la bande passante.

$$\sum_{i=1}^{N} f_i^a \leq f_A. \tag{2.5}$$

Où f_i^a est le taux de traitement attribué à l'utilisateur i qui est limité par le taux de traitement total f_A au niveau du CAP.

$$x_{ij}^l + x_{ij}^a + x_{ij}^c = 1. \tag{2.6}$$

Cette contrainte exige qu'un seul parmi $x_{i,j}^l$, $x_{i,j}^a$ et $x_{i,j}^c$ pour la tâche j de l'utilisateur i peut être 1.

Ce travail va être considéré comme une solution Baseline.

2.3.3 Impact de déchargement sur les ressources utilisées

Parmi les conditions qui ont favorisé l'approche de déchargement, on cite : les ressources disponibles des équipements comme le CPU, le RAM et l'état du réseau.

Un programme linéaire entier est utilisé pour partitionner des tâches en vue de leur exécution entre les serveurs et les nœuds intégrés, dans le but de réduire la combinaison de bande passante réseau et la consommation CPU (Newton *et al.,* 2009).

Des nœuds fog ont été utilisés pour fournir des services de calcul proche d'un équipement d'extrémité et minimiser le temps de réponse de ces nœuds sous une contrainte d'efficacité énergétique (Xiao & Krunz, 2017). Dans ce contexte, un mécanisme distribué est proposé pour l'allocation des ressources informatiques. Ce dernier décide pour chaque nœud fog la partie de la charge de travail de calcul qui nécessite d'être déchargée sur les serveurs infonuagiques distants.

Afin de minimiser le coût total de l'énergie, les retards et les calculs pour tous les utilisateurs mobiles, un mécanisme efficace de déchargement des calculs multi-utilisateurs est proposé qui vise à minimiser le coût de chaque utilisateur mobile individuellement en allouant un canal sans fil à chaque tâche de calcul à décharger (Chen et al., 2016). L'allocation des ressources de communication est censée être suffisante pour les périphériques du réseau.

2.4 Distribution des flux

Pour une application IdO composée des multiples périphériques, l'abondance des données à envoyer à l'infonuagique pour traitement peut saturer la bande passante du réseau et par la suite empêcher l'évolutivité de l'application. Ces auteurs (Moraes de Carvalho et al., 2018) ont posé le sujet de l'impact de la répartition de la charge de travail dans une application IdO en évaluant la possibilité d'augmenter les taux de traitement en exploitant à chaque fois des processeurs de nœud de périphériques plus puissants. Dans ce contexte, ils ont construit une application distribuée qui gère la communication et le traitement à travers des machines de l'infonuagique et des processeurs périphériques en répartissant la charge entre les nœuds infonuagiques et les périphériques afin d'accélérer le traitement par rapport à une application IdO entièrement hébergée en nuage.

2.5 Discussion

Dans cette partie, on va collecter les différents travaux antérieurs qu'on les trouve pertinents et importants pour cerner les limites existantes et amener une contribution dans un tel do-

maine de recherche. Par conséquent, le tableau suivant présente l'importance et les limitations de ces recherches connexes.

Tableau 2-2 Tableau comparatif des travaux connexes

Catégorie	Description	Limitations
Déchargement, énergie consommée et ressources utilisées	- Considérer une application mobile comme une séquence des tâches exécutables en collaboration ente le périphérique mobile et l'informatique clone. (Wen Y 2015) - Décharger les tâches sur l'infonuagique afin de minimiser la consommation énergétique de l'appareil mobile et prolonger la durée de vie de batterie tout en respectant leur délai de traitement. (Wen Y 2015) - Un algorithme branch and bound est proposé pour résoudre le problème d'optimisation et minimiser la consommation d'énergie du calcul et le coût de communication des données pour l'exécution d'une application mobile divisée en des tâches client et serveur. (Zhang et al., 2015) - Des nœuds fog sont utilisés pour fournir des services de calcul proche d'un équipement d'extrémité et minimiser le temps de réponse des nœuds sous une contrainte d'efficacité énergétique (Y. Xiao et M. Krunz.,2017).	- Le coût de communication n'est pas pris en considération .
Distributions des tâches	-Une application distribuée est construite pour gérer la communication et le traitement en répartissant la charge entre les nœuds machines sur le nuage et les périphériques afin d'accélérer le traitement par rapport à une	-Les contraintes de ressources comme RAM et CPU ne sont pas pris en considération.

Tableau 2-2 Tableau comparatif des travaux connexes (suite)

	application IdO entièrement hébergée en nuage. (Moraes de carvalho et al., 2018) - Un schéma d'attribution de poids orienté vers l'énergie coté terminal mobile est proposé. (Ahn et al., 2017) - Concevoir un modèle de déchargement et distribution des tâches vers un cloudlet ou une infonuagique centrée sur l'utilisateur mobile pour réduire les coûts énergétiques, résoudre le problème de la surcharge et améliorer les capacités informatiques. - Des appareils mobiles multiples (tâches multiples) sont pris en considération.	-La supposition d'une bande passante élevée entre les terminaux mobiles engendre une surcharge du système. -Une meilleure stabilité est assurée dans le nuage mais un retard important est détecté.
Déchargement hybride	- Un cadre de travail HyFog et un algorithme de graphes à trois couches sont proposés pour le déchargement des tâches hybrides dans le fog computing qui donne à l'utilisateur mobile le choix d'exécuter localement une tâche, le déchargement D2D et l'exécution dans un fog et le déchargement et l'exécution sur le nuage. -L'objectif est de minimiser le coût total d'exécution de la tâche. (Chen et al., 2017)	- La solution proposée ne s'adapte pas avec un système distribué -Seulement le nombre des cycles CPU est pris en considération pour représenter la contrainte des ressources.
Déchargement trois tiers **Baseline**	Proposé un algorithme MUMTO-C qui vise à optimiser conjointement les décisions de déchargement des tâches et l'allocation de ressources de calcul et de communication pour toutes les tâches dans un système composé des multi-utilisateurs	-Le délai strict des tâches n'est pas pris en considération. -Juste la capacité de CPU est considérée.

Tableau 2-2 Tableau comparatif des travaux connexes (suite)

	mobiles, multitâches, un CAP et un serveur infonuagique. -Le problème est formulé sous forme d'un problème de programmation linéaire en nombre entier dont l'objectif est de minimiser le coût total (Chen et al., 2018)	-Un déchargement à trois tiers est considéré où il y a un seul CAP partagé et un serveur infonuagique

La conclusion tirée des efforts de recherche existants est que la majorité a considéré un seul client comme une solution de déchargement des calculs. Même si ces solutions prennent en compte les modèles de coûts pour la consommation d'énergie et d'autres facteurs temporels (par exemple, le support de communication, la bande passante du réseau et le modèle de programmation) dans le processus de prise de décision pour l'attribution des tâches, elles ne peuvent pas être adoptées comme une solution réaliste.

Aucun travail existant et aucun algorithme proposé ne prend en considération toutes ces métriques de performances comme : les ressources utilisées, l'énergie consommée et le respect du temps d'exécution d'une tâche.

En plus, un autre inconvénient majeur est que l'utilisation d'un modèle informatique distribué n'est pas trop adapté sur tout que le scénario réaliste exige ce genre de modèle où chaque équipement du système a des limites de ressources et il peut être surchargé.

Sans oublier l'hétérogénéité d'un tel système IdO composé de différents composants avec différentes caractéristiques de côté matériel et logiciel qui exige l'autonomie des équipements et en même temps leurs collaborations à fournir des services adaptés au domaine de l'IdO (tenant l'exemple d'une maison intelligente).

Dans notre travail, nous avons modélisé le problème de distribution des tâches comme un problème d'optimisation sous forme de programmation linéaire en nombres entiers mixtes avec variables binaires (MILP:Mixed Interger Linear programming). Nous avons pris en considération toutes les limitations discutées précédemment.

CHAPITRE 3

MÉTHODOLOGIE

3.1 Introduction

Dans ce chapitre, nous décrivons notre système, nous présentons notre solution et notre méthodologie expérimentale qui permettent de répondre à la problématique de notre projet de recherche ainsi que les objectifs présentés dans le premier chapitre. À cet effet, nous exposons les algorithmes proposés. Nous clôturons ce chapitre par un diagramme d'activité.

3.2 Description du système

La figure 3-1 schématise les différents microservices distribués sur les trois environnements de travail (plateformes) dans le cas de demande d'un service de control de la température.

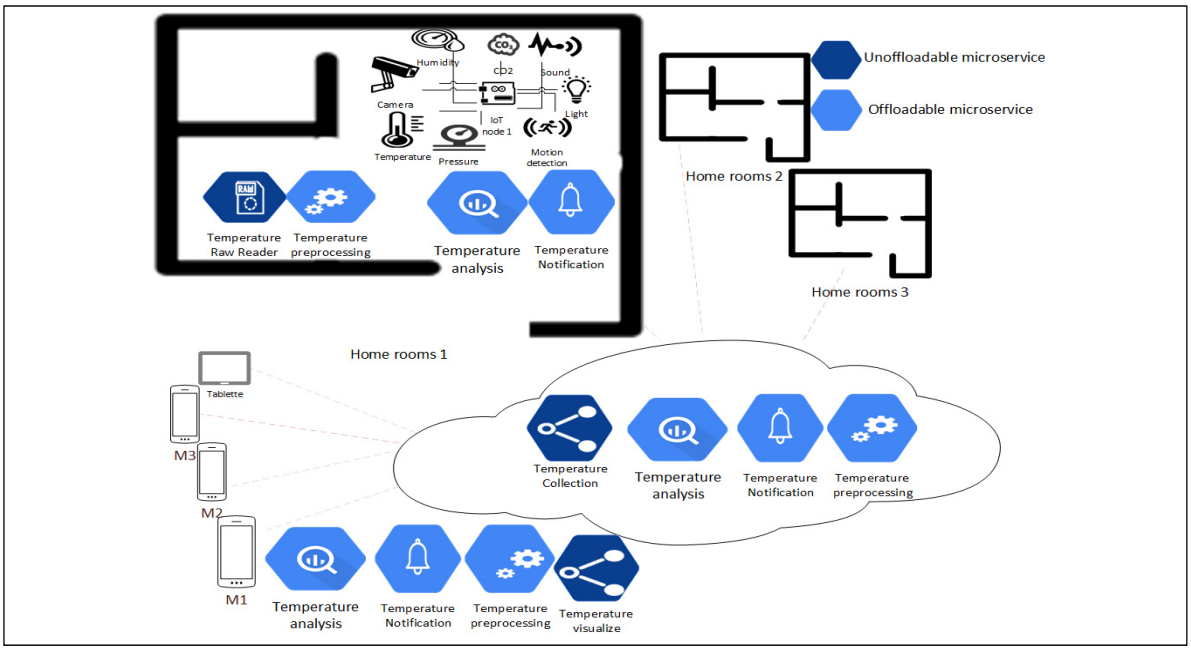

Figure 3-1 Application IdO basée sur des microservices distribués dans un milieu hétérogène

Un fournisseur des services pour les maisons intelligentes souhaite fournir des nœuds IdO et une application mobile qui donne l'accès aux utilisateurs pour contrôler leurs maisons. Chaque nœud IdO va être installé dans une chambre, équipé de différents capteurs et destiné aux habitants de la maison. Les services sont accessibles par chaque membre de la famille.
Disons que notre système est composé d'un ensemble des équipements *Eq* : *N* nœuds IdO qui peuvent être une carte ESP8266, Arduino ou Raspberry Pi. *M* appareils mobiles et *C* conteneurs sur l'infonuagique. Chaque équipement est caractérisé par ses ressources disponibles (CPU, RAM), ses niveaux de batteries (pour les nœuds IdO et les appareils mobiles) et d'autres paramètres comme la puissance locale de traitement et la fréquence locale. Nous supposerons que le nombre de capteurs *S* est identique et que chaque capteur offre des données spécifiques liées au type de capteur. Dans notre exemple, nous avons six capteurs (température, humidité, CO2, lumière, bruit, pression). La liaison entre les nœuds IdO, les appareils mobiles et le nuage (les conteneurs) est sans fil (e.g., WiFi ou 3G/4G). Elle est caractérisée par la puissance de transfert et la bande passante entre les équipements communiquant.
L'application IdO est décomposée en un ensemble de tâches indépendantes *I*. Chaque tâche *i* peut être exécutée sur un microservice et elle est caractérisée par sa charge de travail, les données échangées et relatives à la tâche *i* et la quantité de CPU et RAM nécessaire pour son exécution. Ces microservices peuvent être distribués et exécutés sur trois plates-formes différentes: les nœuds IdO, infonuagique (conteneurs) et les appareils mobiles. Un microservice qui n'exécute aucune tâche peut être désactivé pour économiser la consommation de ressources et d'énergie. Ainsi, chaque tâche est caractérisée par son état déchargeable et non déchargeable présenté respectivement dans la figure 3-1. Une tâche non déchargeable doit être exécutée localement en raison de besoins techniques, tels qu'un capteur relié au nœud IdO dont ses tâches ne peuvent pas être déchargées dans un environnement distant. Tandis qu'une tâche déchargeable a la possibilité d'être traitée localement ou à distance (sur le périphérique IdO, l'appareil mobile ou sur le conteneur sur l'infonuagique) en fonction du contexte et des besoins du système. La distribution des tâches et la décision de les traiter localement sur le nœud IdO, de les décharger et les traiter dans le conteneur sur l'infonuagique

(c'est-à-dire activer / désactiver un microservice) ou sur les appareils mobiles dépendent de certaines contraintes et de l'état du système qui seront traités ultérieurement.

3.3 Solution proposée

La figure 3-2 suivante présente la solution proposée appelée 'le module de suivi' pour résoudre un tel problème ainsi que les différents modules impliqués.

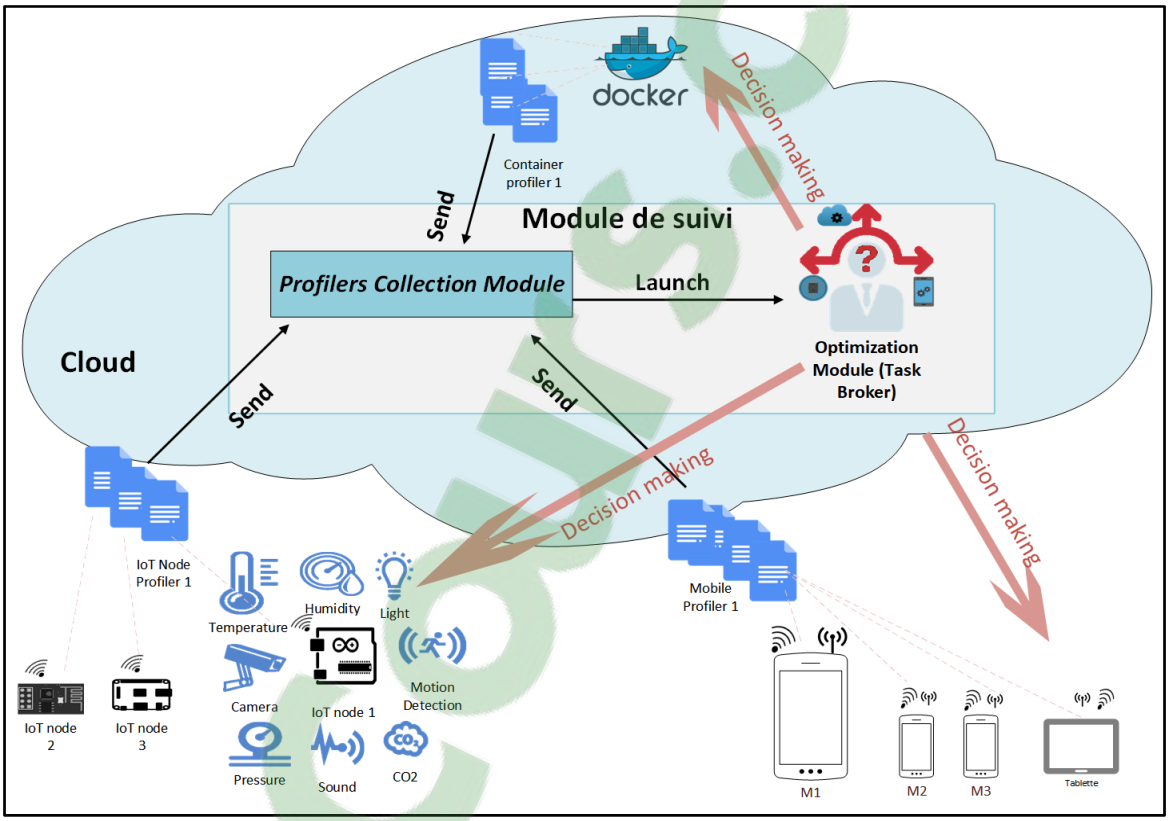

Figure 3-2 Architecture du système

3.3.1 Configuration initiale du système

L'application mobile appelée « MyHome » est installée sur chaque appareil mobile des habitants de la maison. Chaque utilisateur doit configurer manuellement l'application en créant un compte. Deux catégories des utilisateurs sont présentes : Le super utilisateur qui a le droit

à ajouter et manipuler les nœuds IdO reliés à son domicile et les utilisateurs normaux qui reçoivent les données de ces nœuds.

3.3.2 Module de collecte de profileurs

Ce module se concentre sur la collecte de tous les profileurs de différents équipements de notre système. En d'autres termes, chaque nœud IdO, appareil mobile ou conteneur sur l'infonuagique envoie ses capacités en CPU et RAM, la capacité et le niveau de la batterie (pour les nœuds IdO et les appareils mobiles) et la bande passante entre deux équipements communicants. La charge de travail, les données échangées, la quantité de RAM et CPU nécessaire pour traiter une tâche bien précise et la taille des données échangées seront estimées en fonction de multiples tests effectués sur différents éléments du système. Ce module est toujours actif et à l'écoute de tout changement qui peut être : ajout d'un nouvel équipement ou changement d'un des paramètres des équipements du système. Une détection d'un état critique de ces paramètres va déclencher le module d'optimisation.

3.3.3 Module d'optimisation

Appelé aussi module de prise de décision globale. Ce module va être déclenché par le module de collecte de profileurs afin de prendre les décisions appropriées. Selon l'état général du système, ce module va distribuer les tâches et activer / désactiver les microservices par le biais d'un script adapteur. En d'autres termes, il prend la décision à un instant *t* d'exécuter la tâche sur un tel équipement, synchroniser avec toutes les entités du système afin d'éviter qu'un autre équipement traite simultanément la même tâche toute en assurant une consommation énergétique globale minimale et une utilisation optimale des ressources.
La prise de décision et la distribution des microservices seront mises à jour à chaque changement critique de système.

Nous supposons que:

- Chaque nœud IdO noté n_1 va fournir *I* tâches à un appareil mobile m_1 . D'où, on va avoir en total *N* ×*M*× *I* tâches. Puisque tous les équipements IdO vont fournir les mêmes services, ils ont les mêmes tâches à décharger avec des capacités et ressources différentes.
- Certaines tâches doivent être traitées localement, également appelées tâches non déchargeables. En d'autres termes, dans le nœud IdO, dans un conteneur ou dans un appareil mobile. Ces tâches ne seront pas prises en considération dans notre formulation mathématique. On va se concentrer juste sur les tâches déchargeables.
- Toutes les tâches sont indépendantes, ce qui signifie que tous les microservices sont indépendants.
- Chaque équipement ne doit pas surcharger ses capacités des ressources (CPU, RAM).
- Les valeurs des paramètres suivants sont connues préalablement: la charge de travail des tâches *I*, l'utilisation en RAM et CPU de chaque tâche *i*, la taille des données échangées d'une tâche *i*.

3.4 Formulation du problème

Dans cette section, nous formulons le problème d'optimisation. Il consiste à répartir et attribuer dynamiquement les tâches disponibles aux microservices tout en satisfaisant la capacité en ressources de chaque équipement de système, telles que les batteries des nœuds IdO et appareils mobiles, la capacité RAM et CPU, ainsi que l'état de la bande passante.

Afin de minimiser l'énergie globale consommée qui peut être l'énergie de traitement local et l'énergie de transfert, on note *Eq* l'ensemble des équipements composés de *N* nœuds IdO, de *C* conteneurs et de *M* appareils mobiles. Chaque équipement est caractérisé par sa puissance de traitement local (P_n, P_m, P_c), sa fréquence de traitement (S_n, S_m, S_c) et ses ressources $(CPU_n, CPU_m, CPU_c, RAM_n, RAM_m, RAM_c)$. Le nœud IdO est caractérisé par la capacité de sa batterie Cap_n , tandis que l'appareil mobile est caractérisé par sa capacité de la batterie Cap_m et son niveau actuel $Level_m$. Pour la liaison sans fil entre le conteneur et le nœud IdO et l'appareil mobile et le nœud IdO, on note respectivement $P_{tr}(n, c)$,

$P_{tr}(n,m)$, $Bw_{tr}(n,c)$, $Bw_{tr}(n,m)$, la puissance de transfert et de la bande passante entre les équipements: nœud IdO et conteneur, nœud IdO et appareil mobile.

L'application principale sera divisée en plusieurs tâches $i \in I$, chaque tâche est caractérisée par sa charge de travail C_i, ses données D_i, ses ressources nécessaires CPU_i , RAM_i pour exécuter la tâche *i* et son délai limite Δ_i .

D'où, en fonction des capacités de l'équipement et des contraintes du système, nous pouvons formuler notre problème comme un problème de décision avec des variables binaires $\{x_{i,n,m}, y_{i,n,m}, z_{i,n,m}\}$, ou $x_{i,n,m}$, $y_{i,n,m}$ et $z_{i,n,m}$ désignent si la tâche *i* provenant de l'équipement *n* et destinée à l'équipement *m* est traitée respectivement sur l'équipement la plateforme des nœuds IdO n, la plateforme des appareil mobile m ou la plateforme *des* conteneur *c* dans l'infonuagique. Les différents paramètres sont résumés dans le tableau 3-1.

3.4.1 Fonction objective

La fonction objective cherche à minimiser la consommation énergétique globale de notre système composé de nœuds IdO, des appareils mobiles et des conteneurs en affectant équitablement les tâches appropriées à l'équipement approprié tout en respectant le délai limite pour l'exécution et fournissant le service sans drainer les batteries des appareils mobiles, des nœuds IdO et surcharger les ressources (CPU et RAM).

$$\min_{\{x_{i,n,m}, y_{i,n,m}, z_{i,n,m}\}} F = \sum_{i \in I} \left[\sum_{n \in N} \sum_{m \in M} \sum_{c \in C} (G x_{i,n,m} + H y_{i,n,m} + K z_{i,n,m}) \right] \qquad (3.1)$$

S.T. (3.2), (3.3), (3.4), (3.5), (3.6), (3.7), (3.8), (3.9), (3.10), (3.11), (3.12), (3.13), (3.14), (3.15), (3.16), (3.17)

L'ensemble de variables de décision peut être écrit sous la forme d'une matrice M*x* de décision illustrée ci-dessous. Ou les lignes | *Eq* | représentent l'ensemble des équipements (*N, M*

et *C*) et les colonnes |*I*×*N*×*M*| Représentent l'ensemble des tâches provenant de *N* nœuds IdO destinés aux *M* appareils mobiles.

Tableau 3-1 Résumé des notations

Symbole	***Définition***
Eq	Ensemble total des équipements
N, M, C	Ensemble des équipements IdO, appareils mobiles et conteneurs
I	Ensemble des tâches à décharger
$x_{i,n,m}, y_{i,n,m}, z_{i,n,m}$	Les variables de décision pour exécuter respectivement la tâche *i* provenant de l'équipement *n*, destinée à l'équipement *m* dans les plateformes des équipements *n, m* ou *c.*
$E^{n}_{i,n,m}$, $E^{m}_{i,n,m}$, $E^{c}_{i,n,m}$	L'énergie dissipée pour exécuter respectivement la tâche *i* provenant de l'équipement *n*, destinée à l'équipement *m* dans l'équipement *n, m* ou *c.*
$E_{tr}(n,m), E_{tr}(n,c)$	L'énergie consommée lors de communication entre l'équipement *n* et *m*, et l'équipement *n* et *c*
P_n , P_m, P_c	La puissance de traitement local des équipements *n, m* et *c.*
$P_{tr}(n,m), P_{tr}(n,c)$	La puissance de transfert respectivement entre l'équipement *n* et *m*, et entre l'équipement *n* et *c.*
C_i	La charge de travail générée de la tâche *i*
S_n , S_m , S_c	La fréquence d'exécution des équipements *n*, *m* et *c*.
D_i	Les données échangées relatives à la tâche *i*
$Bw_{tr}(n,m), Bw_{tr}(n,c)$	La bande passante entre les équipements *n* et *m* et les équipements *n* et *c*.
Cap_n , Cap_m	La capacité de batterie des équipements *n* et *m*
$Level_m$	Le niveau de batterie actuel de l'équipement *m*
CPU_i	La capacité de CPU nécessaire pour exécuter la tâche *i*

Tableau 3-1 Résumé des notations (suite)

CPU_n , CPU_m , CPU_c	La capacité de CPU dans les équipements *n, m* et *c.*
RAM_n , RAM_m , RAM_c	La capacité de RAM dans les équipements *n, m* et *c.*
RAM_i	La capacité de RAM nécessaire pour exécuter la tâche *i.*
Δ_i	Délai limite pour exécuter la tâche *i.*

Puisque notre objectif principal est de minimiser la consommation totale d'énergie. Les formules suivantes servent à calculer la consommation d'énergie $E^n_{i,n,m}$, $E^m_{i,n,m}$ et $E^c_{i,n,m}$ dans chaque équipement du système.

$$E^n_{i,n,m} = P_n \frac{C_i}{S_n}$$

$$E^m_{i,n,m} = P_m \frac{C_i}{S_m}$$

$$E^c_{i,n,m} = P_c \frac{C_i}{S_c}$$

La décision de décharger la tâche soit vers le conteneur infonuagique ou vers l'appareil mobile inclut non seulement l'énergie consommée dans l'équipement où on décharge, mais aussi l'énergie dissipée lors de transport de la quantité de données entre les deux équipements communicants. D'où, l'énergie de déchargement est définie par l'énergie de traitement dans l'équipement m ou c ($E^m_{i,n,m}, E^c_{i,n,m}$) et l'énergie de transfert entre ces deux équipements ($E_{tr}(n,m), E_{tr}(n,c)$).

$$E_{tr}(n,m) = P_{tr}(n,m) \frac{Di}{Bw_{tr}(n,m)}$$

$$E_{tr}(n,c) = P_{tr}(n,c) \frac{Di}{Bw_{tr}(n,c)}$$

$$G = E^n_{i,n,m} \tag{3.2}$$

$$H = (E^m_{i,n,m} + E_{tr}(n,m)) \tag{3.3}$$

$$K = (E^c_{i,n,m} + E_{tr}(n,c)) \tag{3.4}$$

3.4.2 Contraintes

3.4.2.1 Distribution des tâches

La contrainte suivante dicte qu'un seul équipement peut exécuter la même tâche à un instant *t*. Elle est considérée nécessaire pour garantir la synchronisation de tous les équipements de notre système, et par la suite assurer une utilisation optimale des ressources des équipements.

$$x_{i,n,m} + y_{i,n,m} + z_{i,n,m} = 1, \quad \forall\, i \in I, \forall\, n \in N, \forall\, m \in M, \forall\, c \in C \tag{3.5}$$

3.4.2.2 Ressources des équipements

L'un des objectifs qu'on veut atteindre est d'utiliser les différentes ressources des équipements de façon raisonnable. C'est pour cela, la prise de décision de déchargement est liée aux ressources disponibles dans chaque équipement ainsi que les ressources nécessaires pour exécuter une tâche bien précise. D'où, les contraintes suivantes assurent que l'ensemble des ressources nécessaires pour exécuter localement les tâches ne doivent pas dépasser la capacité des ressources existantes.

$$\sum_{i \in I} \sum_{m \in M} RAM_i\, x_{i,n,m} \;\leq\; RAM_n\,, \quad \forall\, n \in N \tag{3.6}$$

$$\sum_{i \in I} \sum_{m \in M} CPU_i\, x_{i,n,m}, \leq\; CPU_n\,, \quad \forall\, n \in N \tag{3.7}$$

$$\sum_{i \in I} \sum_{n \in N} RAM_i\; y_{i,n,m} \;\leq\; RAM_m\;, \quad \forall\, m \in M \tag{3.8}$$

$$\sum_{i \in I} \sum_{n \in N} CPU_i\; y_{i,n,m} \;\leq\; CPU_m\,, \quad \forall\, m \in M \tag{3.9}$$

De côté infonuagique, cette contrainte est plus importante, vu qu'une surcharge d'utilisation des ressources de conteneur peut amener des pénalités aux utilisateurs.

$$\sum_{n \in N} \sum_{m \in M} RAM_i \, z_{i,n,m} \leq RAM_c, \qquad \forall i \in I, \forall c \in C \tag{3.10}$$

$$\sum_{n \in N} \sum_{m \in M} CPU_i \, z_{i,n,m} \leq CPU_c, \qquad \forall i \in I, \forall c \in C \tag{3.11}$$

3.4.2.3 Capacité des batteries

L'un des points faibles des nœuds IdO et des appareils mobiles est leurs capacités de batterie. C'est pour cela, on définit ces contraintes où l'ensemble de l'énergie consommée sur un équipement *n* ou *m* ne doit pas dépasser sa capacité actuelle de batterie Cap_n et Cap_m.

$$\sum_{i \in I} \sum_{m \in M} P_n \frac{C_i}{S_n} x_{i,n,m} \leq Cap_n, \qquad \forall n \in N \tag{3.12}$$

De côté appareil mobile, à instant *t*, l'équipement *m* peut avoir un niveau de batterie qui varie entre [1%, 100%]. Au contraire des nœuds IdO, les appareils mobiles ont d'autres processus qui tournent en arrière-plan et qui peuvent avoir une consommation d'énergie importante. Afin d'éviter l'état critique de la batterie, le niveau de la batterie $Level_m$ doit être entre cet intervalle [30%,100%].

$$\sum_{i \in I} \sum_{n \in N} \left(P_m \frac{C_i}{S_m} + P_{tr}(n,m) \frac{D_i}{BW_{tr}(n,m)} \right) y_{i,n,m} \leq Level_m \times Cap_m, \tag{3.13}$$

$$\forall m \in M$$

3.4.2.4 Délai limite des tâches

Assurer que chaque tâche est exécutée dans son délai limite est assez important pour fournir une meilleure qualité d'expérience aux utilisateurs. Pour l'équipement *n*, le temps d'exécution d'une tâche *i* ne doit pas dépasser le délai défini. Pour les équipements *m* et *c*, le temps d'exécution d'une tâche *i* et le temps de transfert des données relatives à l'exécution de cette tâche doivent être inférieurs ou égaux au délai limite pour fournir le service relié à cette tâche.

$$\frac{C_i}{S_n}\, x_{i,n,m}, \leq\ \Delta_i, \qquad i \in I, n \in N\ , \forall\ m \in M \tag{3.14}$$

$$\left(\frac{C_i}{S_m} + \frac{D_i}{BW_{tr}(n,m)}\right)\ y_{i,n,m} \leq\ \Delta_i\ , \qquad \forall\ i \in I\ , \forall\ n \in N\ , \forall\ m \in M \tag{3.15}$$

$$\left(\frac{C_i}{S_c} + \frac{D_i}{BW_{tr}(n,c)}\right) z_{i,n,m} \leq\ \Delta_i\ , \qquad \forall\ i \in I\ , \forall\ n \in N\ , \forall\ m \in M\ , \forall\ c \in C \tag{3.16}$$

3.5 Les algorithmes proposés

Dans cette section, nous présentons les différents algorithmes proposés pour résoudre notre problème. Ainsi, nous faisons la comparaison avec le résultat obtenu par le solveur: la méthode génération de coupe ('cut-generation') et la méthode heuristique.

3.5.1 Fonction Intlinprog du MATLAB

La fonction ou le solveur de MATLAB intlinprog se base sur l'algorithme dual simplex. C'est une technique en programmation linéaire pour résoudre un problème d'optimisation.
Elle implique une fonction et plusieurs contraintes exprimées en inégalités. Les inégalités définissent une région polygonale, et la solution se situe généralement à l'un des sommets.
L'algorithme dual simplex est le plus adapté aux problèmes pour lesquels une première solution réalisable double est facilement disponible. Il est particulièrement utile pour réoptimiser un problème après l'ajout d'une contrainte ou la modification de certains paramètres afin que la base précédemment optimale ne soit plus possible. Le principe de fonctionnement : il sé-

lectionne d'abord une variable pour quitter la base, puis trouve la variable qui doit entrer dans la base pour maintenir une double faisabilité.

3.5.2 Méthode génération de coupe

La méthode génération de coupe est proposée pour résoudre les problèmes liés aux MILP. Elle est considérée comme une approche alternative de la méthode des banches et des liaisons (Branch and Bound). Cette méthode fonctionne de la manière suivante (figure 3.3) : telle que la région réalisable sera considérée comme un plan. Afin de se rapprocher et de détecter la solution optimale entière, une coupe sera appliquée pour réduire la taille du plan avec des contraintes supplémentaires. L'intlinprog de MATLAB fournit cette méthode comme option. L'application des générations de coupe (méthode de coupe du plan) se caractérise par sa rapidité de résolution du problème. Telle qu'elle peut accélérer la vitesse de calcul de la solution.

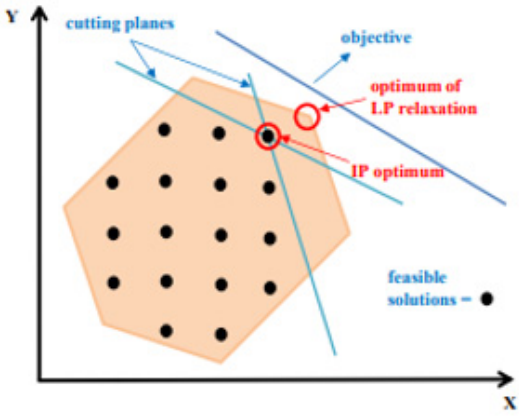

Figure 3.3 Méthode de coupe du plan (Punyisa et al., 2018)

3.5.3 Méthode heuristique

Elle utilise la méthode diviser et conquérir. Cette dernière consiste à diviser le problème en instances de problème plus petites et de les résoudre. Ensuite, combinez les résultats. Cette

technique cherche à trouver les limites supérieures sur la fonction objective plus rapidement (Taner,2016, p. 10). Dans notre cas, on va utiliser l'option 'round' de la méthode heuristique du solveur intlinprog de MATLAB. Ce solveur prend la solution LP au problème relâché d'un nœud. Il arrondit les composants entiers de manière à maintenir la faisabilité.

3.5.4 Algorithme 1 : DT_IoTMC

Cet algorithme est appelé DT_IoTMC ('Distribute Tasks between IoT nodes, Mobile phones and containers on Cloud') qui signifie les tâches distribuées entre les nœuds IdO, les conteneurs sur l'infonuagique et les appareils mobiles. Cet algorithme adapte le type de déchargement hybride. Il est développé pour ressourdre la fonction objective (3.1) mentionnée dans le chapitre précèdent.

Algorithme 3.1 Algorithme DT-IoTCM

Input: *N, M, C, I,* Tasks characteristics, Tasks resource consumption, equipment profilers
Output: Distributed matrix *X* and optimal overall energy consumption *F*

Initialize $F = 0$, $X_{i,n,m} = \emptyset$, $Y_{i,n,m} = \emptyset$, $Z_{i,n,m} = \emptyset$, $Mx = \emptyset$;
While ($F = 0$) **do**
  Calculate *G, H, K* according to (2-4)
  Set objective function f according to G, *H, K*
  Set A_{ineq} and b_{ineq} according to constraints (6-16)
  Set A_{eq} and b_{eq} according to constraint (5)
  Set lb, ub and $intcon$
  $[x, F] = intlinprog(f, intcon, A_{ineq}, b_{ineq}, A_{eq}, b_{eq}, lb, ub)$
  $X_{i,n,m} = reshape(x, N, I \times N \times M)$
  $Y_{i,n,m} = reshape(x, M, I \times N \times M)$
  $Z_{i,n,m} = reshape(x, C, I \times N \times M)$
  $Mx = \{X_{i,n,m} Y_{i,n,m}, Z_{i,n,m}\}$
end

3.5.5 Algorithme 2 : DT_IoTM

Cet algorithme est appelé DT_IoTM (‘Distribute Tasks between IoT nodes and Mobile phones’) qui signifie les tâches distribuées entre les nœuds IdO et les appareils mobiles. On a développé cet algorithme pour prouver l’efficacité de notre solution initiale en comparant les deux résultats obtenus. Ce dernier sert à traiter les tâches localement ou les décharger vers les appareils mobiles, appelé aussi déchargement à deux niveaux. Une simple modification doit être apportée à la contrainte (3.5) puisqu’on ne va pas considérer le déchargement vers l’infonuagique.

$$x_{i,n,m} + y_{i,n,m} = 1, \quad \forall\, i \in I, \forall\, n \in N, \forall\, m \in M \tag{3.17}$$

Algorithme 3.2 Algorithme DT-IoTM

Input: *N, M, I,* Tasks characteristics, Tasks resources consumption, equipment profilers
Output: Distributed matrix *X* and optimal overall energy consumption *F*

Initialize $F = 0$, $X_{i,n,m} = \emptyset$, $Y_{i,n,m} = \emptyset$, $Mx = \emptyset$;
While ($F = 0$) **do**
    Calculate *G, H* according to (2), (3)
    Set objective function f according to G, *H*
    Set A_{ineq} and b_{ineq} according to constraints (6-9) and (12-15)
    Set A_{eq} and b_{eq} according to constraint (17)
    Set lb, ub and $intcon$
    $[x, F] = intlinprog(f, intcon, A_{ineq}, b_{ineq}, A_{eq}, b_{eq}, lb, ub)$
    $X_{i,n,m} = reshape(x, N, I \times N \times M)$
    $Y_{i,n,m} = reshape(x, M, I \times N \times M)$
    $Mx = \{X_{i,n,m}, Y_{i,n,m}\}$
end

Pour tous les algorithmes précédents, les entrées sont le nombre des équipements du système (nœuds IdO, conteneurs et appareils mobiles), les tâches à décharger et leurs capacités néces-

saires en ressources (RAM et CPU) ainsi que les profileurs des équipements participants. Ces derniers contiennent les caractéristiques de chaque équipement (CPU, RAM, capacité des batteries, les niveaux des batteries des appareils mobiles, les bandes passantes réelles entre les équipements). Les sorties vont être la matrice décisionnelle Mx et l'énergie totale consommée $\mathcal{F}$.

Tout d'abord, on doit initialiser les variables de sorties à zéro (ligne 1). Tant que la consommation totale en énergie est encore nulle, on énonce la fonction objective (lignes 3-4), on prépare les matrices des contraintes d'inégalités et d'égalités (lignes 5-6), on exige que toutes les variables soient de types binaires et qu'elles puissent être 0 ou 1 (ligne 7). Ensuite, on appelle notre solveur avec tous les paramètres nécessaires (ligne 8). Puis, on remodèle nos matrices $X_{i,n,m}, Y_{i,n,m}$, et/ou $Z_{i,n,m}$. Enfin, on combine les dernières matrices pour avoir la matrice décisionnelle Mx qui contienne les emplacements convenables des tâches.

3.5.6 Solution Baseline

Dans cette partie, on va expliquer brièvement la solution récente (Chen et al.,2018) qui s'approche de notre travail. Elle consiste à minimiser le coût total pondéré de l'énergie, des calculs dans un système informatique composé de plusieurs utilisateurs, un CAP partagé et un serveur infonuagique distant, en décidant le déchargement à trois niveaux des tâches dépendantes de chaque utilisateur et l'allocation des ressources de communication.
Les auteurs ont calculé les limites inférieures de coût énergétique minimal pour les considérer par la suite comme une référence d'évaluation des performances.
L'algorithme MUMTO-C est proposé pour calculer la solution optimale locale qui va être comparer aux limites inferieures.
Notre solution peut être considérée d'un certain point, une amélioration de la solution MUMTO-C. Puisque dans notre modélisation, on a pris en considération certaines perspectives proposées par ses auteurs. Comme les délais stricts des tâches et l'adaptation d'un scénario qui utilise plusieurs points de déchargement. En plus, Chen et al ont utilisé la méthode de déchargement à trois tiers avec un point d'accès partagé entre tous les utilisateurs alors

que dans notre solution on va suivre le déchargement hybride où le déchargement va être vers différents containeurs et appareils mobiles.

CHAPITRE 4

RÉSULTATS EXPÉRIMENTAUX

4.1 Introduction

Dans ce chapitre, nous préparons tout d'abord notre système composé de trois plateformes différentes. : Plateforme des nœuds IdO, plateforme nuage contenant les microservices nécessaires hébergés dans des conteneurs, et l'application mobile 'MyHome'. Ensuite, nous implémentons le système et ses modules proposés au niveau de la méthodologie. Puis, nous étudions un scénario et nous tirons les résultats et les interprétations. Enfin, nous faisons une discussion afin de souligner les objectifs achevés.

4.2 Implémentation du système

Dans le chapitre méthodologie, l'architecture du système est présentée. Dans cette section, on va détailler son implémentation.

Afin de mettre en place notre système et tester notre modèle.

- Nous avons conçu une plateforme équipée des nœuds IdO. La programmation de ces nœuds est faite en suivant l'architecture microservices.
- Nous avons développé une application mobile intitulée 'MyHome'. Cette application suit l'architecture des microservices et elle va être installée sur tous les appareils mobiles qui forment la deuxième plateforme de notre système.
- La troisième plateforme sur l'infonuagique est composée des conteneurs Docker. Les microservices vont être lancés dans des conteneurs.
- Les même microservices reliés aux tâches déchargeables (Tableau 1-1) sont présents dans les trois plateformes impliquées dans notre système. Les autres microservices non déchargeables sont aussi présents dans les plateformes appropriées.

4.2.1 Plateforme des nœuds IdO

Dans les travaux antérieurs faits dans le domaine IdO, la source de données était toujours considérée comme une boite noire. Ils n'ont pas pris en considération les contraintes qui peuvent exister dans les cas réels comme la capacité de la batterie d'un nœud IdO, et la capacité de ses ressources.

Afin de combler cette faille, on a développé des nœuds IdO. On a utilisé Arduino Uno et ESP8266 comme des nœuds processeurs. Arduino Uno est une carte électronique développée par Arduino.cc et basée sur le microcontrôleur ATmega328P. Cette carte n'est pas équipée par un module Wifi c'est pour cela on lui a attaché un ESP8266.

ESP8266 est développé par le fabricant chinois Espressif. Il est un circuit intégré équipé par un microcontrôleur programmable avec une connexion WiFi. Il est très utilisé dans le domaine IdO grâce à son bas prix. Ses ressources sont considérées importantes par rapport à Arduino Uno.

Ces nœuds sont équipés par un ensemble de capteurs : un capteur de température, un capteur d'humidité, un capteur de lumière, un capteur de pression, un capteur du son et un capteur de CO2. Afin de programmer ces microcontrôleurs (nœuds IdO), on a utilisé l'Arduino IDE. On a respecté l'architecture des microservices dans notre programmation de façon où chaque type de capteur soit isolé et fonctionne indépendamment des autres. Cela favorise l'évolutivité du projet et l'ajout des autres capteurs dans le futur. Une autre caractéristique importante de cette architecture est qu'elle facilite l'intervention pour localiser et corriger un problème matériel ou logiciel.

Comme détaillé dans la section 1.2 et mentionné dans le tableau 1-1 et la figure 3-1, la tâche non déchargeable reliée à cette plateforme et le microservice qui lui correspond est : data_raw_reader. D'autres microservices développés sont considérés importants: IoT_profile_microservice qui collecte les caractéristiques du nœud IdO comme la capacité de la batterie, les capacités RAM et CPU.

Le IoT_Managment_microservice qui collecte l'adresse MAC du nœud ainsi que son emplacement et utilise l'API REST pour ajouter le nœud IdO dans la base de données relative et recevoir en retour son Id. Tous ces microservices sont implémentés dans chaque nœud IdO.
La figure 4-1 schématise un nœud IdO.

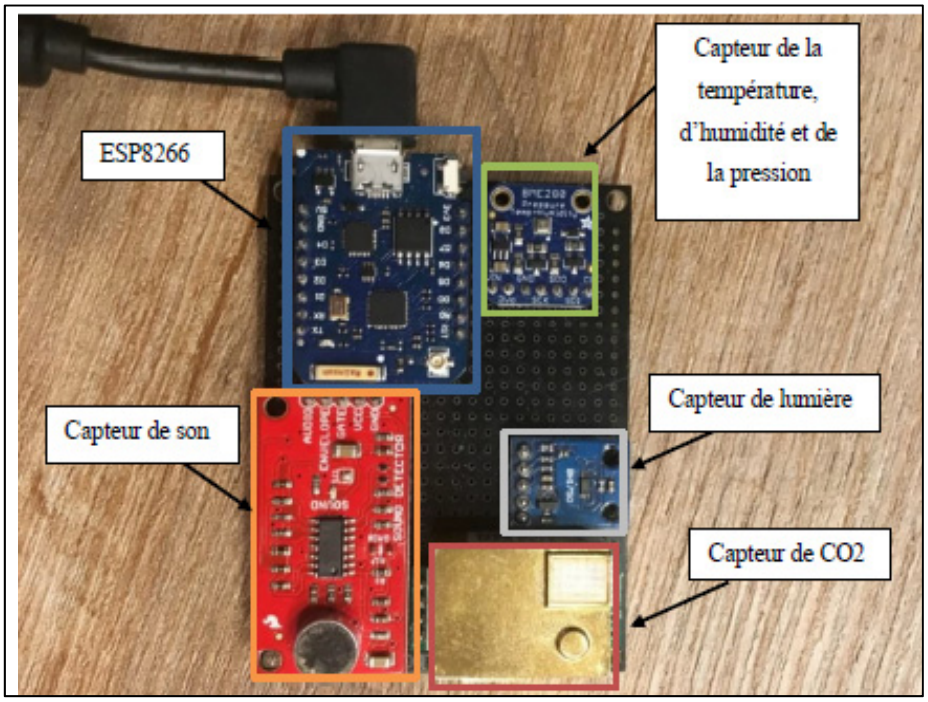

Figure 4-1 Un ESP8266 équipé par différents capteurs (nœud IdO)

Les données sont envoyées périodiquement, chaque 5 secondes à un courtier Mosquitto situé sur l'infonuagique en utilisant le protocole MQTT. Ce dernier est un protocole de messagerie très utilisé dans les applications IdO qui se base sur TCP/IP. Son principe de fonctionnement est basé sur Publish/Subscribe qui représentent les clients fournisseurs et demandeurs d'un service au près d'un courtier MQTT comme Mosquitto.
La figure 4-2 suivante schématise les données interceptées au niveau d'un nœud IdO de type ESP8266. Ces données vont être par la suite publier sur le courtier Mosquitto.

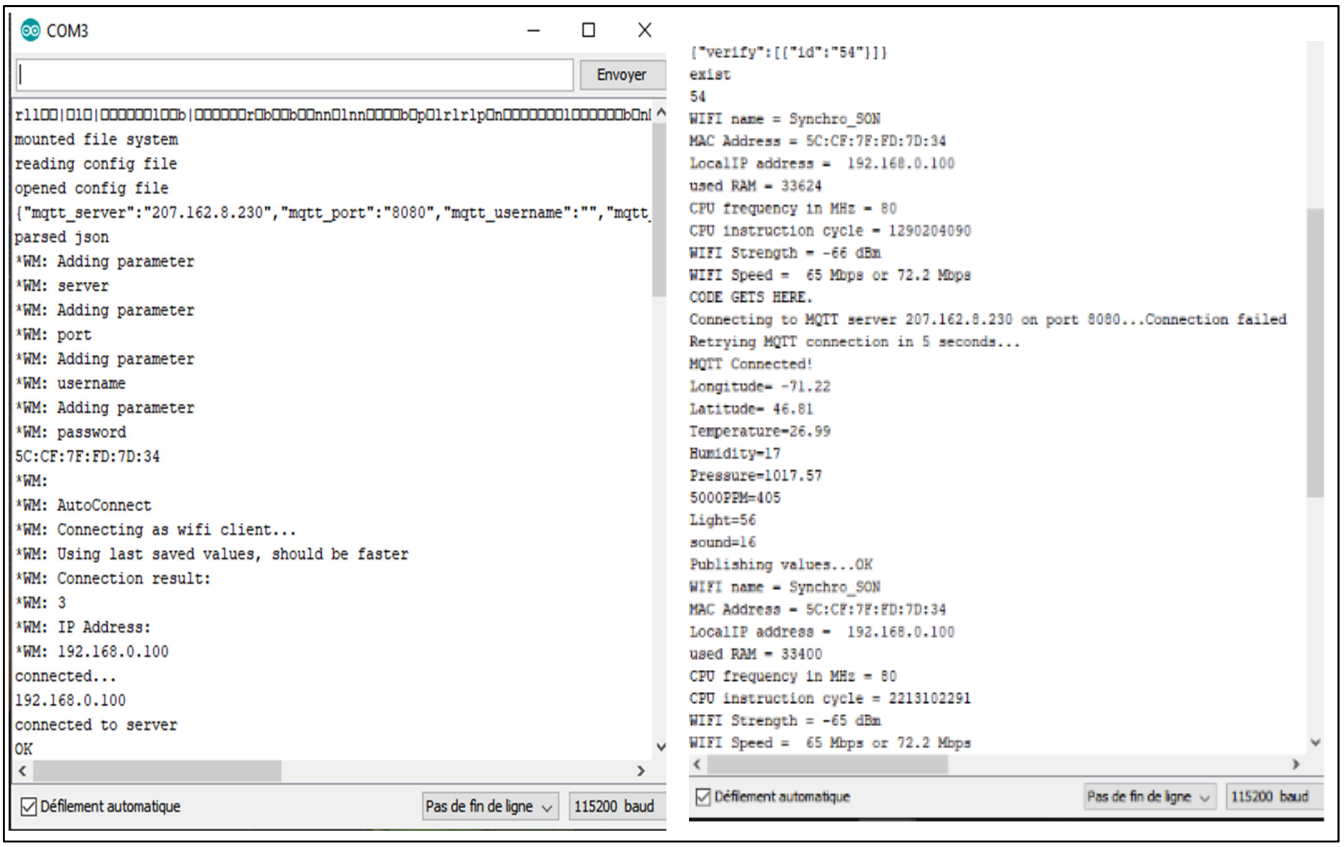

Figure 4-2 Les données interceptées au niveau un nœud IdO de type ESP8266

La figure 4-3 montre les différents nœuds captés au niveau de courtier MQTT (Mosquitto Broker) présent sur l'infonuagique.

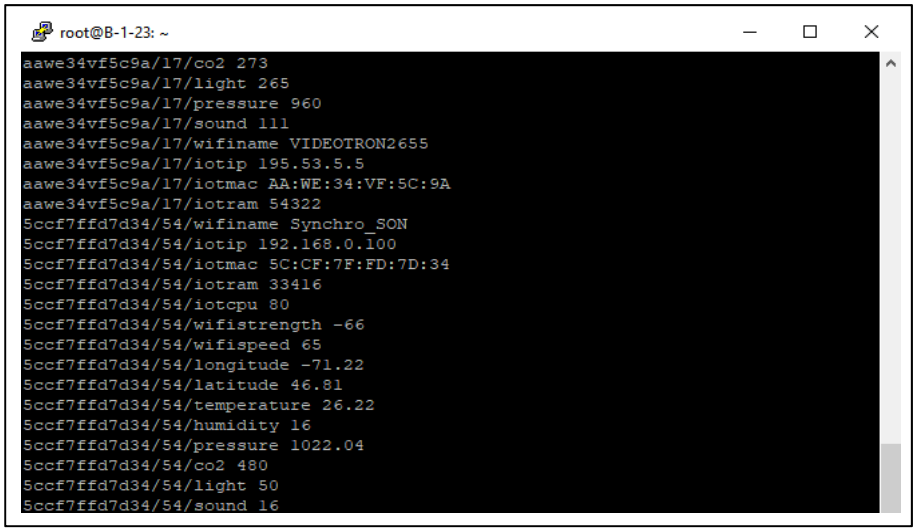

Figure 4-3 Les données interceptées au niveau de courtier Mosquitto

4.2.2 Plateforme des appareils mobiles (l'application mobile 'MyHome')

Cette application est destinée à tous les appareils mobiles qui utilisent Android comme système d'exploitation. Toute personne qui veut contrôler sa maison et recevoir les informations relatives à son état en temps réel doit créer un compte 'MyHome'. Au moins, un utilisateur peut être considéré un 'super_user'. Ce dernier a le droit de configurer entrer les nœuds IdO disponible dans la maison dont il veut recevoir les données. Une fois le nœud IdO est affecté à l'utilisateur, ce dernier va recevoir tous les services disponibles relatifs aux différents capteurs présents comme contrôle de la température, contrôle d'humidité….

Étant donné qu'il a tous les privilèges, cet utilisateur peut affecter des nœuds aux autres utilisateurs normaux, c.-à-d., les autres habitants de la maison.

Même l'application mobile suit l'architecture des microservices. Elle contient aussi les mêmes microservices que dans les nœuds IdO et infonuagique.

Les figures suivantes montrent les différentes interfaces de notre application 'MyHome'.

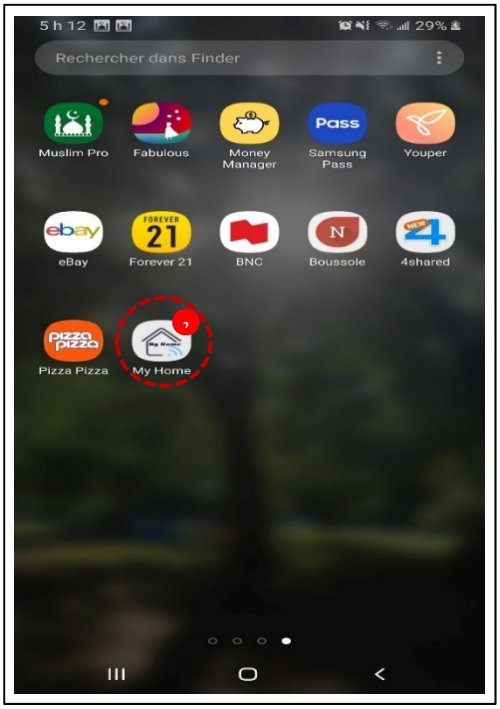

Figure 4-4 L'application 'MyHome'

Figure 4-5 Interface principale de l'application

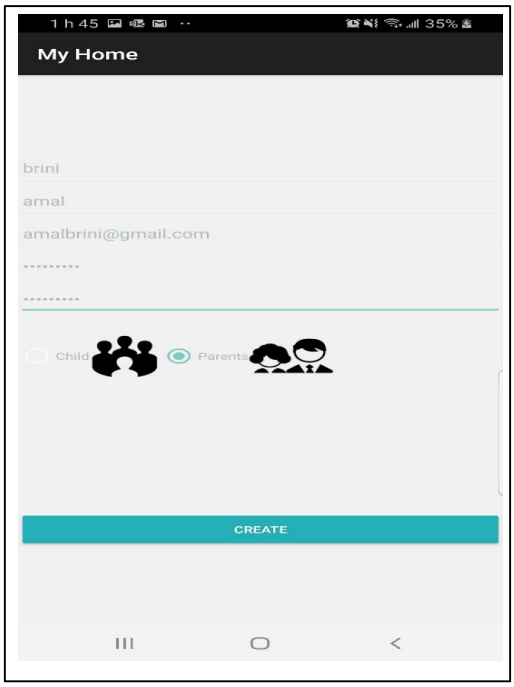

Figure 4-6 Interface ‘Création d’un compte’

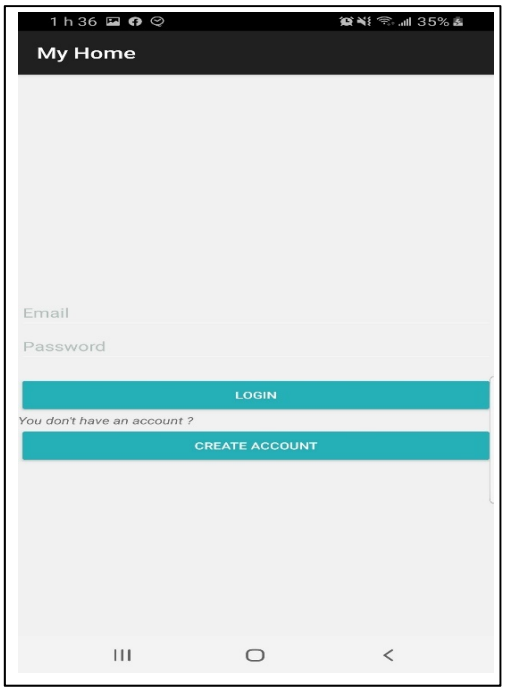

Figure 4-7 Interface ‘Connexion’

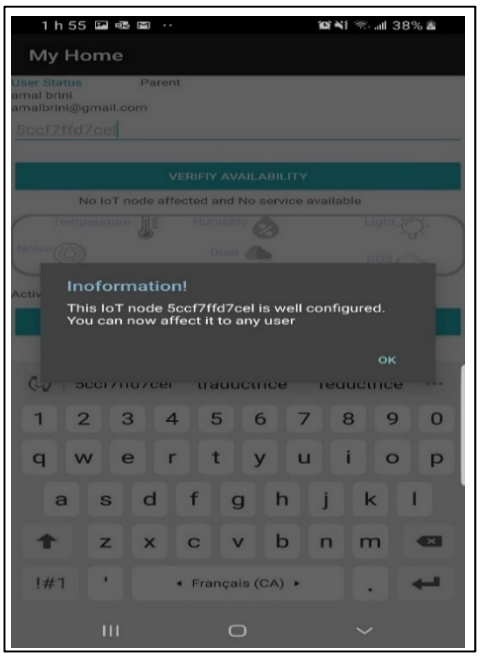

Figure 4-8 Interface ‘Configuration d’un nœud IdO’

Figure 4-9 Interface ‘Suivi des données des capteurs en temps réel’

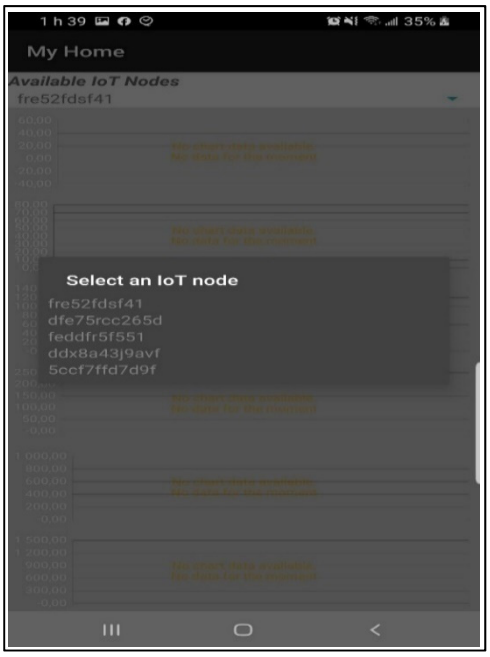

Figure 4-10 Interface ‘Choisir un nœud pour visualiser ces données’

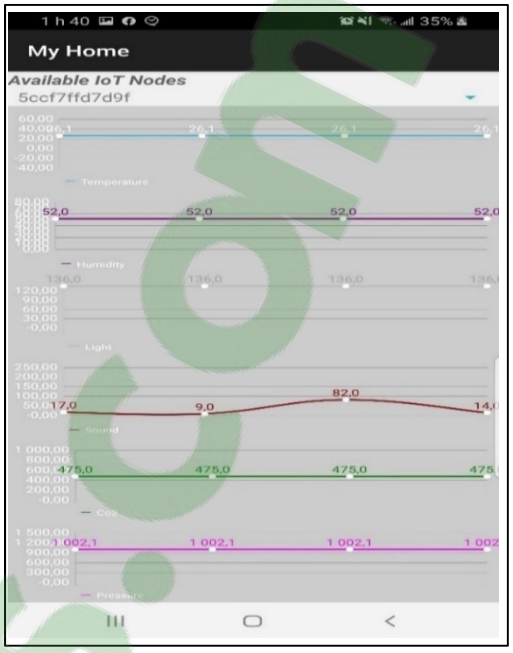

Figure 4-11 Interface ‘Visualiser toutes les données d’un nœud IdO’

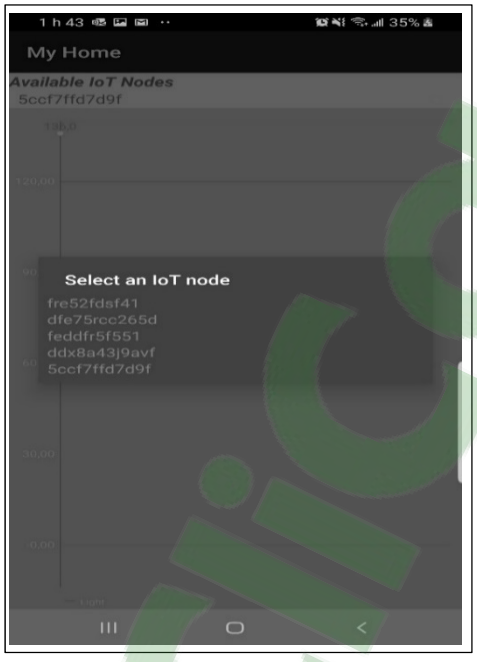

Figure 4-12 Interface ‘Choisir un capteur d’un nœud bien précis

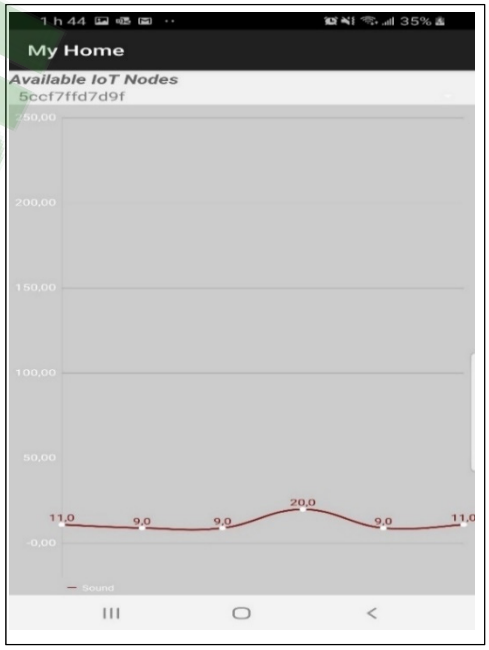

Figure 4-13 Interface ‘Visualiser les données d’un capteur relié à un nœud choisi

4.2.3 Plateforme sur l'infonuagique (les conteneurs)

Sur l'environnement de travail infonuagique, on a développé les microservices reliés aux tâches déchargeables mentionnées dans la section 1-2. Comme expliqué, ces microservices ou conteneurs vont être actifs selon la décision prise par notre module d'optimisation. À part de nos modules, un script adapteur responsable de l'activation et désactivation des microservices sur les différentes plateformes vont être implémenter dans l'infonuagique.

4.3 Protocole de validation

Le module d'optimisation décrit dans la section 3.3.3 va contenir nos algorithmes. Afin de valider notre solution qui est le déchargement hybride en utilisant les trois plateformes du système, on a comparé les résultats obtenus de nos algorithmes proposés DT-IoTM et DT-IoTMC dans des scénarios différents. Comme on a eu recours à la solution Baseline pour surligner les améliorations détectées. On a reproduit le même scénario de la solution Baseline avec les mêmes paramètres dans un scénario où le nombre des nœuds IdO et le nombre des appareils mobiles sont égaux à 5 et que le nombre des tâches déchargeables par nœud IdO varie entre cet intervalle [4, 6, 8, 10]. Le résultat de notre solution est la valeur minimale de l'énergie consommée par le système au complet ainsi qu'une matrice de distribution des tâches entre les différents équipements (les nœuds IdO, les conteneurs et les appareils mobiles) des trois plateformes impliquées.

Le protocole de validation suit les étapes suivantes :

- Implémenter et tester la performance des algorithmes développés DT_IoTMC et DT_IoTM en fonction de la consommation totale d'énergie du système global et de la consommation énergétique de chaque équipement du système après la distribution des tâches dans chaque plateforme.
- Comparer les performances de nos algorithmes en utilisant le MILP solveur, la méthode heuristique et la méthode coupure du plan en termes de la consommation totale d'énergie du système au complet.

- Comparer les performances en termes de la consommation totale d'énergie des solutions des algorithmes et la solution Baseline.
- Adopter différents scénarios en variant à chaque fois un des paramètres système comme le nombre des tâches à décharger, le nombre des nœuds IdO et le nombre des appareils mobiles. Vérifier leurs impacts sur la consommation totale d'énergie du système entier et la consommation énergétique de chaque équipement actif du système.

4.4 Diagramme d'activité global

La figure 4-14 suivante schématise le fonctionnement de notre système en global. Du point de configuration de l'application mobile jusqu'à la réception des données relatives à l'application IdO du contrôle domestique.

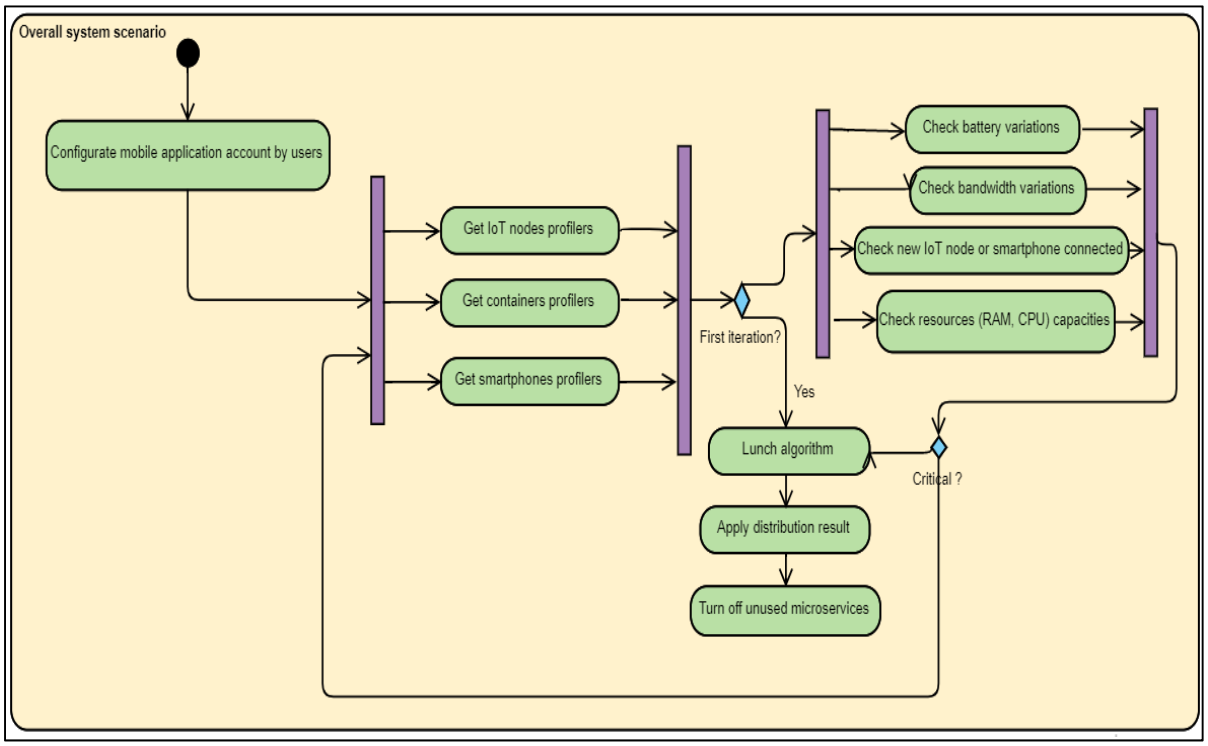

Figure 4-14 Diagramme d'activité pour la solution proposée

Tout d'abord, il faut préparer les différentes plateformes impliquées afin de faire fonctionner l'application du contrôle de la maison intelligente. Dans ce contexte, les nœuds IdO doivent

être connectés au réseau disponible. Ainsi que l'application mobile MyHome doit être installée et configurée sur les appareils mobiles des utilisateurs.
Une fois tous les équipements du système sont connectés, les différents profileurs IdO, des conteneurs et des appareils mobiles collectent périodiquement toutes les informations nécessaires au fonctionnement de notre algorithme. Les profileurs sont des scripts qui tournent toujours en arrière-plan. Par défaut, ils sont initialisés nul. Le passage de valeurs nuls à d'autres valeurs est considéré comme la première itération qui va déclencher à son tour notre algorithme DT_IoTMC. Tous ces scripts de profileurs comparent les valeurs détectées par les capacités des équipements et les valeurs critiques des batteries. C.-à-d., tout changement critique détecté dans ces paramètres : le nombre des nœuds IdO, le nombre des appareils mobiles, les batteries des nœuds IdO et des appareils mobiles, les capacités des ressources des équipements système, l'état actuel du réseau (la capacité d'envoi entre deux équipements) va déclencher l'algorithme. Le résultat de distribution des tâches sera appliqué. Ce qui conduit à son tour à l'activation des microservices impliqués.

4.4.1 Environnement de simulation

Afin de tester et valider notre travail, on va réaliser un ensemble de simulations tout en variant à chaque fois un paramètre système.
La figure 4-15 montre notre environnement du travail où chaque chambre de la maison intelligente est équipée d'un nœud IdO.
Le tableau 4-1 résume nos paramètres et leurs intervalles de valeurs. Ces derniers sont pris des tests faits (comme les capacités de calcul de la charge de travail des tâches, les tailles de données échangées, les capacités des ressources et les délais limites des tâches) et de certains travaux antérieurs (comme les puissances locales de traitement et de transfert).

Figure 4-15 Environnement du travail

Tableau 4-1 Tableau des variables du système

Paramètres	**Valeurs**
Nombre *N* des nœuds IdO	[1-20]
Nombre *M* des appareils mobiles	[1-20]
Nombre *C* des conteneurs	[4-50]
Nombre *I* des tâches à décharger	[4-50]
Puissance locale d'exécution P_n	[0.1W-1W]
Puissance locale d'execution P_m	[0.2W-2W]
Puissance locale d'execution P_c	[0.2W-3W]
Puissance de transfert $P_{tr}(n,m)$ entre *n* et *m*	[0.1W-2W]
Puissance de transfert $P_{tr}(n,c)$ entre *n* et *c*	[0.1W-2W]
C_i	[40-200] Mégacycles

Tableau 4-1 Tableau des variables du système (suite)

Fréquence de calcul de l'équipement S_n	[80MHz-1.4GHz]
Fréquence de calcul de l'équipement S_m	[0.5GHz-1.6GHz]
Fréquence de calcul de l'équipement S_c	[2GHz-5GHz]
Taille de données échangées Di	[100-1000] KB
$Bw_{tr}(n,m), Bw_{tr}(n,c)$	[1-5Mbps]
Capacité de la batterie de noeud $IdO\ n\ Cap_n$	[500J-3000J]
Capacité de la batterie de l'appareil mobile $m\ Cap_m$	[500J-3000J]
Niveau de la batterie $Level_m$ de l'appareil mobile m	[5%,100%]
Capacité en RAM de l'équipement $n\ RAM_n$	[256KB-1GB]
Capacité en RAM de l'équipement $m\ RAM_m$	[500MB-2GB]
Capacité en RAM de l'équipement $c\ RAM_c$	[512MB-3GB]
RAM_i	[128-1024] KB
CPU_i	[0.02%-15%]
Capacité en CPU de l'équipement $n\ CPU_n$	[1-4]
Capacité en CPU de l'équipement m CPU_m	[2-8]
Capacité en CPU de l'équipement $c\ CPU_c$	[2-10]
Délai limite Δ_i d'une tâche	[1s-60s]

4.4.1.1 Évaluation de la consommation d'énergie totale par rapport au nombre des tâches déchargeables

Puisque notre objectif principal est de minimiser la consommation totale de l'énergie de système au complet, on a choisi de fixer le nombre des nœuds IdO, le nombre des appareils mobiles et de faire varier le nombre des tâches à décharger.

Le tableau 4-2 qui suit résume les différentes valeurs choisies pour effectuer ce test :

Tableau 4-2 Configuration de l'environnement du travail

Paramètres	***Valeurs***
N	5
M	5
C	[2,4,6,8,10,12,14,16,18,20]
Tâches	[2,4,6,8,10,12,14,16,18,20]
Ci	[40,60,200,160,100,90,150,110,160,100,100,90,150,110,160,100,40,60,200,160] en Megacycles
P_n	[0.5,0.8,2,1.2,1] en Watt
P_m	[0.7,1,1.1,1,1.2] en Watt
P_c	[0.02,0.08,0.15,0.1,1,0.4,0.7,0.5,0.4,0.7,0.02,0.08,0.4,0.7,0.08,0.15,0.02,0.08 ,0.15,0.1] en Watt
$P_{tr}(n,m)$	[0.4,0.1,0.2,0.8,0.5;0.2,1,1.2,0.05,0.03;0.1,0.3,1,0.5,1;1,0.4,0.05,1.1,0.9; 1.1,0.7,0.05,0.4,0.3] en Watt
$P_{tr}(n,c)$	[1,0.5,1.5,1,1.1] en Watt
S_n	[80,160,1000,900,1000] en MHz
S_m	[0.5,1.2,1.5,1.2,1.6] en GHz
S_c	[2.1,2.5,4,3,2,1.8,3,2.4,1.8,3,2.1,2.5,1.8,3,2.5,4,2.1,2.5,4,3] en GHz
Di	[100,250,850,580,400,330,550,420,400,330,580,400,330,580,850,580,100, 250,750,580] en KB
$Bw_{tr}(n,m)$	[2.2,2,5,3,1.2;2,3,3,5,2.2;2,3,1,5,1.1;2.2,3,1.2,4,3.5;3.5,2,1.5,2.3,2] en Mbps
$Bw_{tr}(n,c)$	[2,1,1.1,2.2,2] en Mbps
cap_n	[500,1000,2500,2000,2200] en Joule
Cap_m	[1000,1500,2000,1200,2500] en Joule

Tableau 4-2 Configuration de l'environnement du travail (suite)

$Level_m$	[50,90,35,40,70] en %
RAM_n	[256,512,16000,900000,256000] en KB
RAM_m	[500,800,2000,1000,1200] en MB
RAM_c	[512,750,1600,1000,3000,1100,1500,2000,3000,1100,2000,3000,1500,2000, 2000,3000,512,750,1600,1000] en MB
RAM_i	[100,210,1000,760,350,300,900,835,350,300,350,300,350,300,760,350,100,210, 1000,760] en KB
CPU_n	[1,1,2,2,2]
CPU_m	[4,4,4,4,4]
CPU_c	[4,4,8,6,7,5,6,6,5,6,8,6,6,7,5,6,4,4,8,6]
CPU_i	[0.09,0.1,4,2,1,0.5,1.5,1.1,0.9,0.6,1.1,0.9,0.9,0.6,0.5,0.5,0.09,0.1,3,2] en %
Δ_i	[0.2,0.4,1,0.8,0.6,0.55,0.7,0.65,0.5,0.4,0.7,0.65,0.7,0.65,1,0.8,0.2,0.4,1,0.8] en s

La figure 4-16 suivante schématise la consommation totale de l'énergie en variant le nombre des tâches à décharger.

D'une autre façon, le nombre des conteneurs varie respectivement dans cet intervalle [2,4,6,8,10,12,14,16,18,20].

On va présenter les résultats obtenus en utilisant les deux algorithmes proposés.

Selon les résultats obtenus, on constate que choisir l'algorithme DT-IoTMC qui consiste à décider de traiter une tâche localement sur l'équipement IdO, ou de choisir de la décharger à un conteneur en nuage ou vers l'appareil mobile qui l'a demandée donne la moindre consommation énergétique totale.

➔Cela explique que l'algorithme DT_IoTMC s'adapte facilement au comportement du système, chaque fois qu'il reçoit une tâche assez lourde, il choisit entre le conteneur infonuagique et l'appareil mobile qui consomme le moins d'énergie.

L'algorithme DT-IoTM qui consiste à traiter la tâche localement ou de la décharger vers l'appareil mobile, présente une consommation d'énergie plus importante par rapport à l'autre algorithme.

➔On remarque que cet algorithme n'a pas d'autre choix que de décharger les tâches lourdes vers les appareils mobiles bien que parfois cette solution n'est pas la meilleure.

➔Chaque fois le nombre des tâches à décharger augmente, la consommation totale d'énergie augmente aussi. D'où, on conclut que la consommation totale d'énergie est proportionnelle au nombre des tâches à décharger.

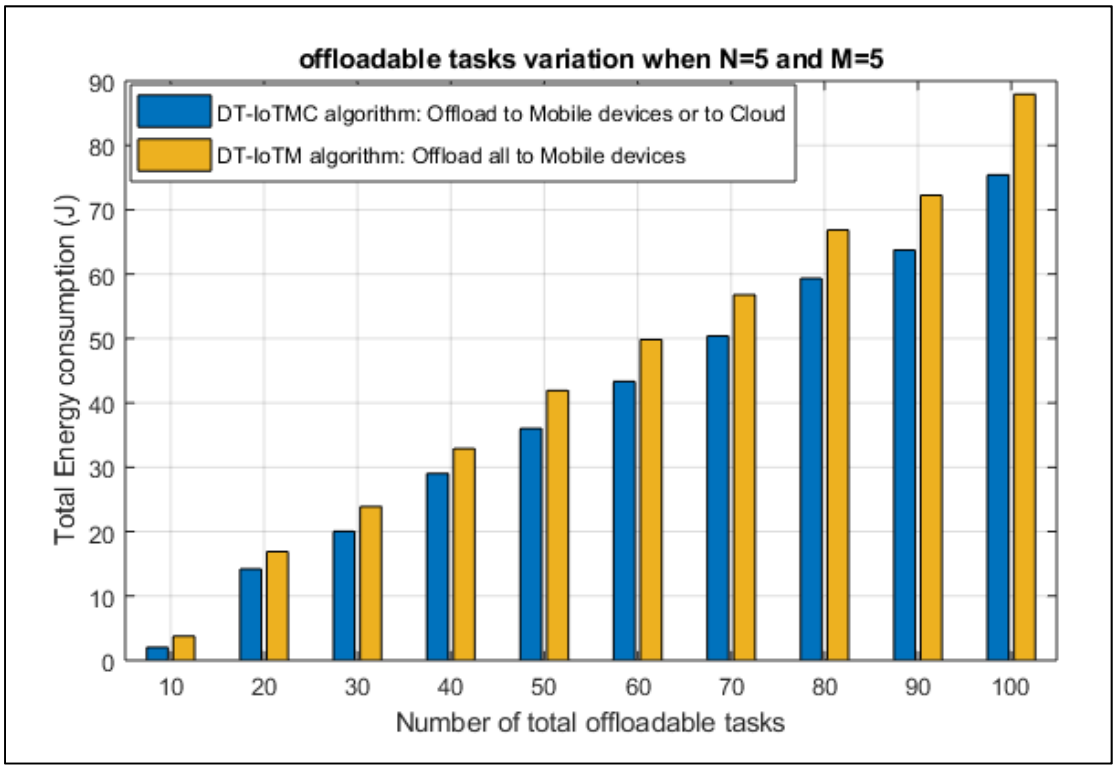

Figure 4-16 Effets de la variation du nombre des tâches déchargeables sur la consommation totale d'énergie quand N=M=5

Le tableau 4-3 présente la consommation totale en énergie pour les différents types d'algorithmes proposés ainsi que les équipements système qui n'ont traité aucune tâche. L'exemple choisi est pour $N = M = 5$, et le nombre des tâches est égale à 4.

On peut tirer que l'algorithme 1 (DT_IoTMC) qui suit un déchargement hybride entre les nœuds IdO, les conteneurs et les appareils mobiles consomment moins d'énergie par rapport

à l'algorithme 2 (DT_IoTM). Cela est dû à la présence des conteneurs infonuagiques. Ces derniers ont traité des tâches considérées lourdes pour les appareils mobiles grâce à la richesse en ressources de l'infonuagique. Ce qui a donné une consommation énergétique faible sur les conteneurs grâce à leurs fréquences de traitement importantes.

Tableau 4-3 La consommation totale d'énergie et les équipements non actifs pour les deux algorithmes proposés

Algorithme	***Consommation totale d'énergie***	***Équipements non actifs***
Algorithme 1 : DT_IoTMC	15.69108 J	n_1, n_2
Algorithme 2 : DT_IoTM	16.209409J	n_1, n_2

➔On remarque que les deux premiers équipements IdO n'ont pas traité des tâches. Ils se sont limités à des tâches non déchargeables. Cela est dû à leurs faibles capacités. Ces algorithmes ont trouvé un état de réseau convenable pour décharger les tâches vers une autre plateforme riche en ressources, précisément une plateforme avec une fréquence de calcul plus importante, qui donne la moindre consommation totale d'énergie.

Les tableaux (tableau 4-4 tableau 4-5) ainsi que la figure 4-17 présentent la consommation énergétique de chaque équipement système pour les différents algorithmes proposés ainsi que les tâches traitées sur chaque équipement quand : $N = M = 5$ et le nombre des tâches égale à 4.

Tableau 4-4 La consommation énergétique de chaque équipement du système et les tâches traitées là-dessus (cas de l'algorithme 1)

Équipements	***Tâches***	***Énergie consommée***
n_3	13,15,38,40,63,65,88,90	1.84J
n_4	41,42,44,45,66,67,69,70,91,94,95	2.0267J
n_5	21,22,24,25,46,47,49,50,71,72,74,75,96,97,99,100	1.84J
m_1	11,31,36,56,61,81,86	1.489J
m_2	2,12,27,37,52,62,77,87,92	1.2443J
m_3	3,18,23,28,43,48,53,68,73,78,93,98	1.2167J
m_4	9,14,29,34,39,54,59,64,79,84,89	1.7605J
m_5	10,35,60,85	0.3693J
c_1	1,4,5,6,7,8,16,17,19,20	0.4856J
c_2	26,30,32,33	0.5077J
c_3	51,55,57,58	1.7300J
c_4	76,80,82,83	1.1813J

Tableau 4-5 La consommation énergétique de chaque équipement du système et les tâches traitées là-dessus (cas de l'algorithme 2)

Équipements	***Tâches***	***Énergie consommée***
n_3	13,15,38,40,63,65,88,90	1.8400J
n_4	16,19,20,41,42,44,45,66,67,69,70,91,94,95	2.1867J
n_5	21,22,24,25,46,47,49,50,71,72,74,75,96,97,99, 100	1.8400J
m_1	1,6,11,26,31,36,51,56,61,76,81,86	2.5226J
m_2	2,7,12,17,27,32,37,52,57,62,77,82,87,92	2.2677J
m_3	3,8,18,23,28,33,43,48,53,58,68,73,78,83,93,98	2.2660J
m_4	4,9,14,29,34,39,54,59,64,79,84,89	1.8205J
m_5	5,10,30,35,55,60,80,85	1.4559 J

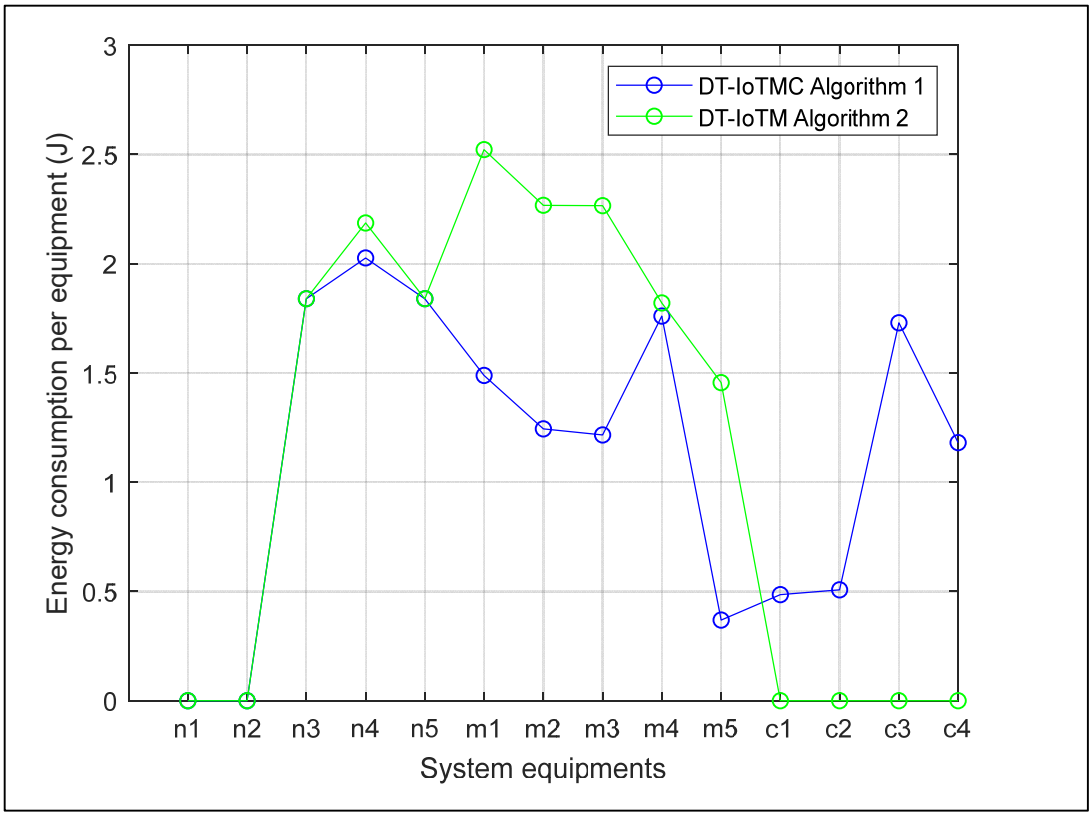

Figure 4-17 La consommation énergétique totale de chaque équipement du système en exécutant les deux algorithmes

Selon la figure 4-17 on remarque que le premier algorithme DT_IoTMC donne des meilleurs résultats par rapport au deuxième algorithme DT_IoTM, vu qu'il distribue les tâches entre les trois plateformes. Comme c'est clair sur la figure, le nœud IdO numéro 4 ainsi que les appareils mobiles 1,2,3 et 5 ont bien profité de la distribution des tâches sur les trois plateformes puisque leurs consommations énergétiques ont bien diminué. Les conteneurs dans le nuage ont pris la relève en traitant les tâches gourmandes en ressources et en énergie. D'où, l'algorithme DT_IoTMC a bien respecté les exigences des applications IdO et il a distribué équitablement la charge de travail et les tâches. Autrement dit, sans surcharger les ressources, sans drainer les batteries et bien sûr en respectant les délais des tâches. Les conteneurs (c1, c2, c3 et c4) sont inactifs et ils ne consomment pas de l'énergie lorsqu'on utilise l'algorithme DT_IoTM.

4.4.1.2 Évaluation de la consommation d'énergie totale par rapport au nombre des nœuds IdO

Afin de voir l'impact d'augmentation de nombre des nœuds IdO sur la consommation totale d'énergie du système, on a décidé de fixer le nombre des appareils mobiles à *M*=10, le nombre des tâches déchargeables par équipement I=10 et de faire varier le nombre des nœuds IdO dans cet intervalle [5,15].

La figure 4-18 suivante évalue la consommation totale d'énergie en exécutant les deux types d'algorithmes proposés.

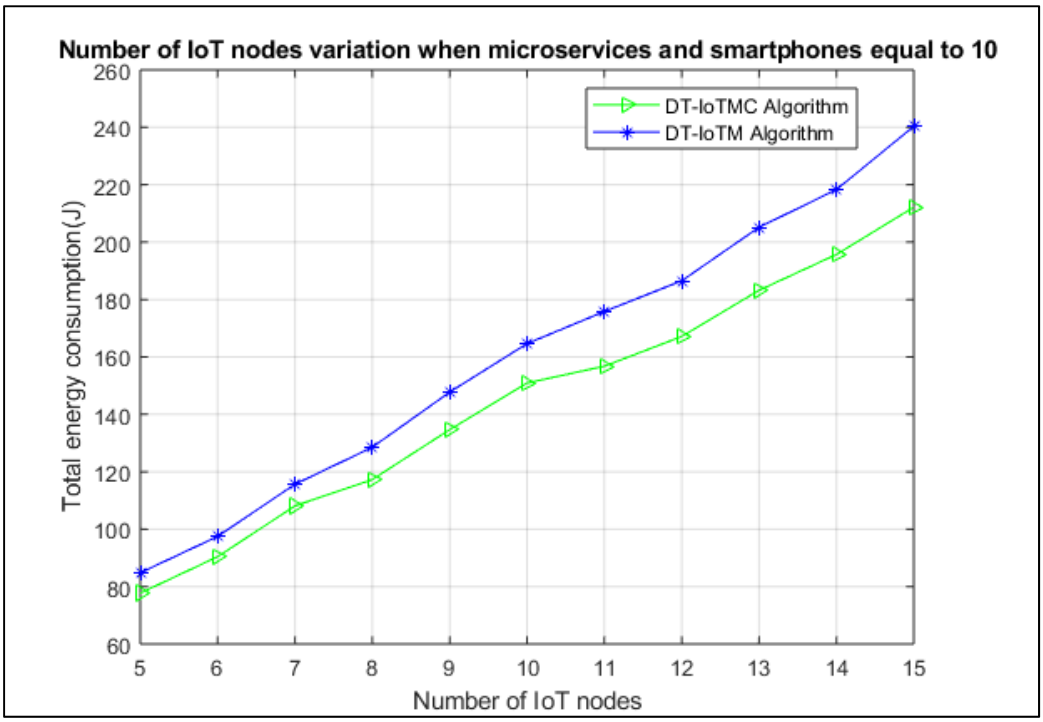

Figure 4-18 L'effet de variation du nombre des nœuds IdO sur la consommation totale de l'énergie quand M et les microservices sont égales à 10

À partir de ce graphe et pour les deux algorithmes, on remarque que la consommation totale d'énergie augmente proportionnellement par rapport au nombre des équipements *N*.

La courbe verte, résultante de l'algorithme DT_IoTMC donne le meilleur résultat (la plus faible consommation totale d'énergie). Cela s'explique par le fait que chaque fois que l'algorithme trouve une bonne qualité de bande passante disponible qui résulte à une faible consommation d'énergie de communication, il favorise le déchargement vers l'infonuagique

que vers l'appareil mobile. L'énergie dissipée en exécution est beaucoup plus importante sur l'appareil mobile que sur l'infonuagique (grande fréquence de calcul de CPU).

La courbe bleue (Algorithme 2) donne une consommation totale d'énergie assez importante qui peut être expliquée par la grande quantité d'énergie consommée lors de déchargement des données et l'exécution des tâches lourdes sur les appareils mobiles. Ainsi que l'absence d'une entité plus puissante et riche en ressources comme l'infonuagique a prouvé son impact sur la consommation totale de l'énergie.

La figure 4-19 schématise la consommation énergétique pour chaque équipement du système ou le nombre des nœuds IdO, les microservices et les appareils mobiles sont fixés respectivement à 5,10 et 10.

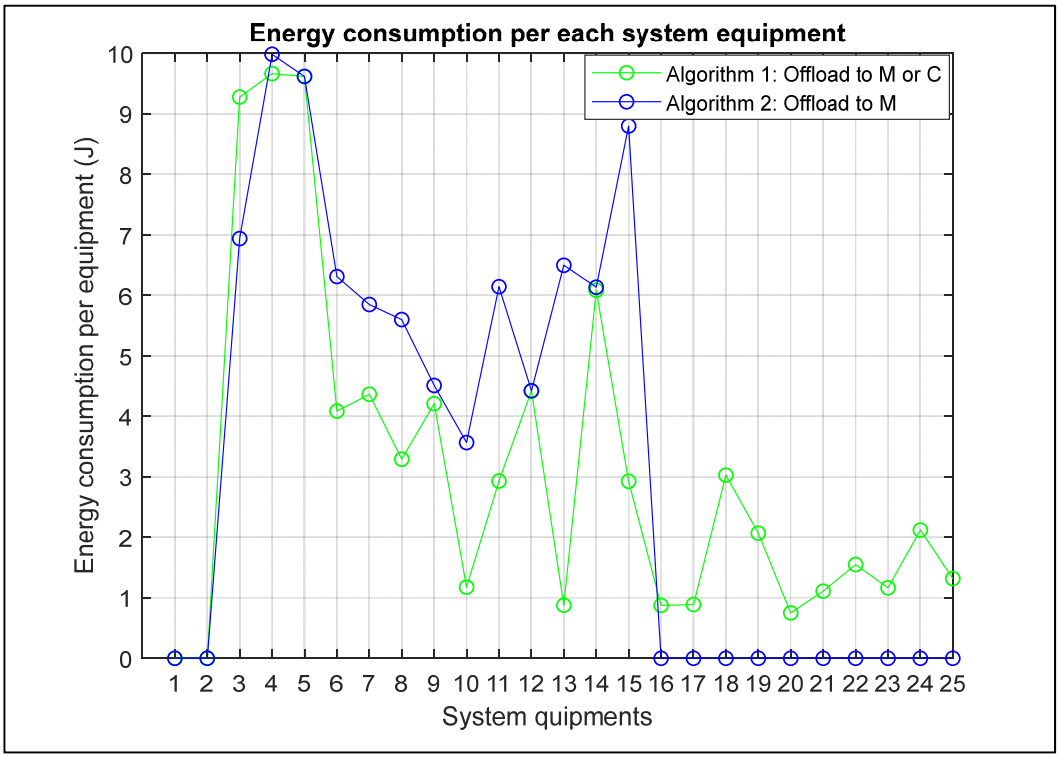

Figure 4-19 La consommation d'énergie de chaque équipement du système

On peut conclure que l'algorithme 1 donne les meilleurs résultats par rapport au deuxième algorithme. Comme remarqué dans l'intervalle [6,15], la consommation d'énergie de chaque

appareil mobile a marqué une diminution dramatique puisque les tâches lourdes sont déchargées vers les conteneurs sur le nuage. D'où, le partage de tâches entre les nœuds IdO, les appareils mobiles et les conteneurs peut donner une durée de vie plus longue aux composants.

La figure 4-20 suivante schématise la consommation totale d'énergie en exécutant l'algorithme DT_IoTMC. La simulation est réalisée avec un nombre fixe de tâches à décharger (I=10), un nombre des appareils mobiles M=10 et un nombre variable des équipements IdO $N = [5,15]$.

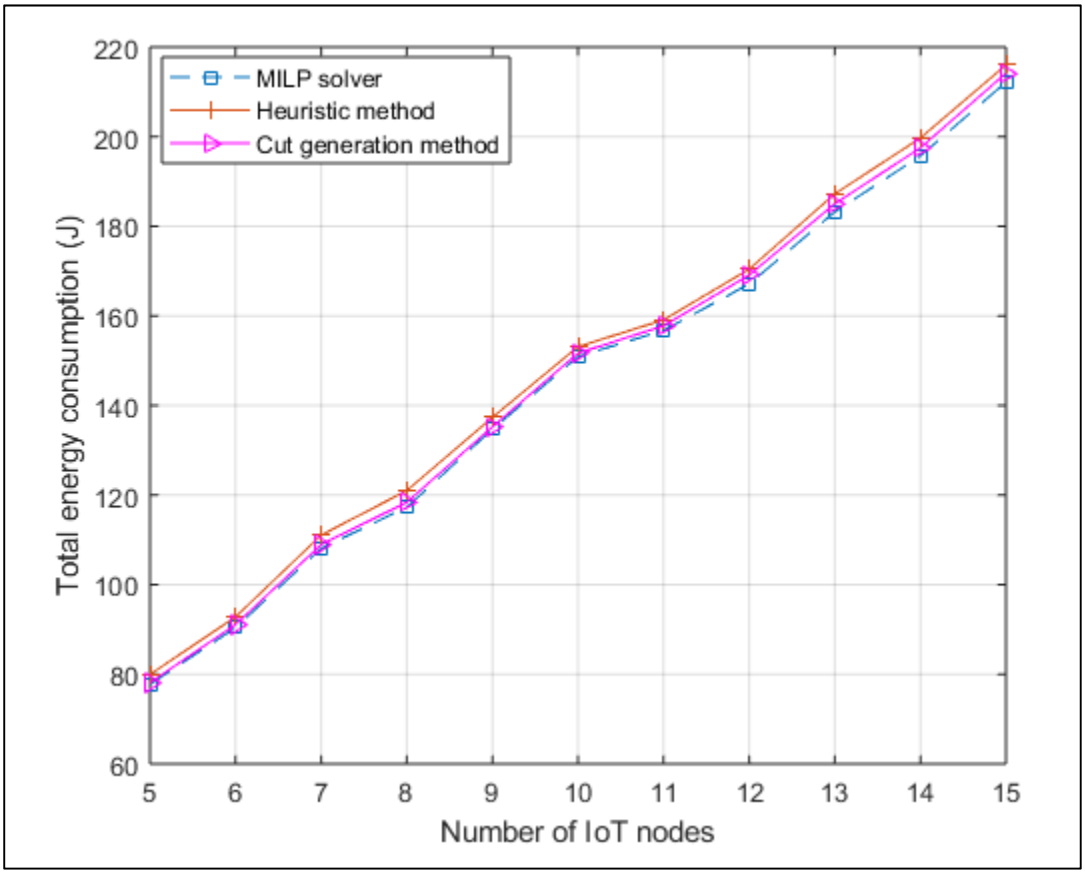

Figure 4-20 La comparaison des résultats de solver MILP avec la méthode de coupure du plan et la méthode heuristique en appliquant l'algorithme DT_IoTMC

Les courbes obtenues sont les suivants : une courbe bleue de solveur, une courbe violette avec la méthode de coupure du plan et une courbe rouge avec la méthode heuristique.

À partir de ce graphe, on tire que la courbe bleue donne la solution optimale. La méthode coupure du plan donne des solutions faisables et proches de l'optimale.

La méthode heuristique donne aussi une solution faisable avec la plus grande consommation d'énergie.

Du côté performance, la méthode coupure du plan réduit considérablement le temps de calcul comparativement à la méthode heuristique.

La figure 4-21 suivante présente les résultats obtenus en exécutant l'algorithme DT_IoTM et les comparent avec les méthodes coupure du plan et heuristique.

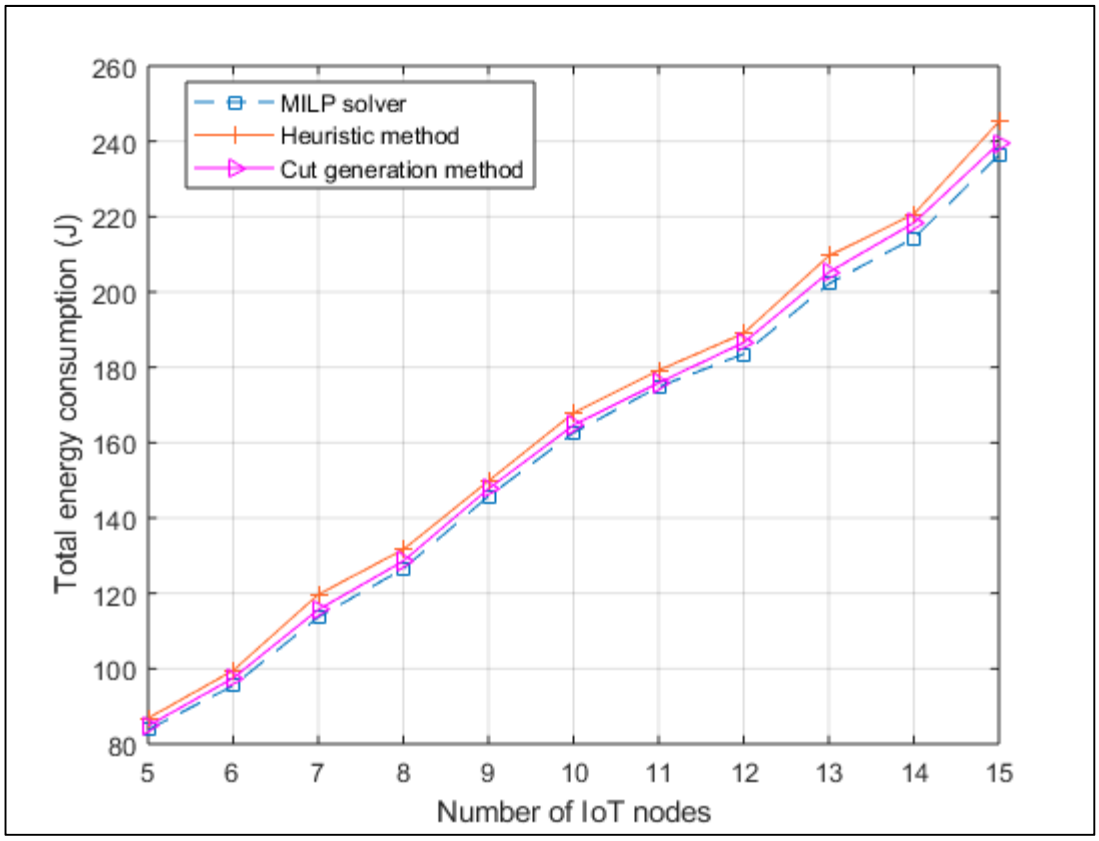

Figure 4-21 La comparaison des résultats de solver MILP avec la méthode de coupure du plan et la méthode heuristique en appliquant l'algorithme DT_IoTM

On remarque que toujours le solveur MILP (la courbe bleue) donne la solution optimale. Alors que la méthode de coupure du plan et la méthode heuristique donne des solutions faisables proches de l'optimale.

4.4.1.3 Évaluation de la consommation d'énergie totale par rapport au nombre des appareils mobiles

Dans cette partie, on va fixer le nombre des microservices et le nombre des équipements IdO à 10. On va varier le nombre des appareils mobiles $e_M = [5,10,15]$ et on va suivre le comportement de nos deux algorithmes proposés.

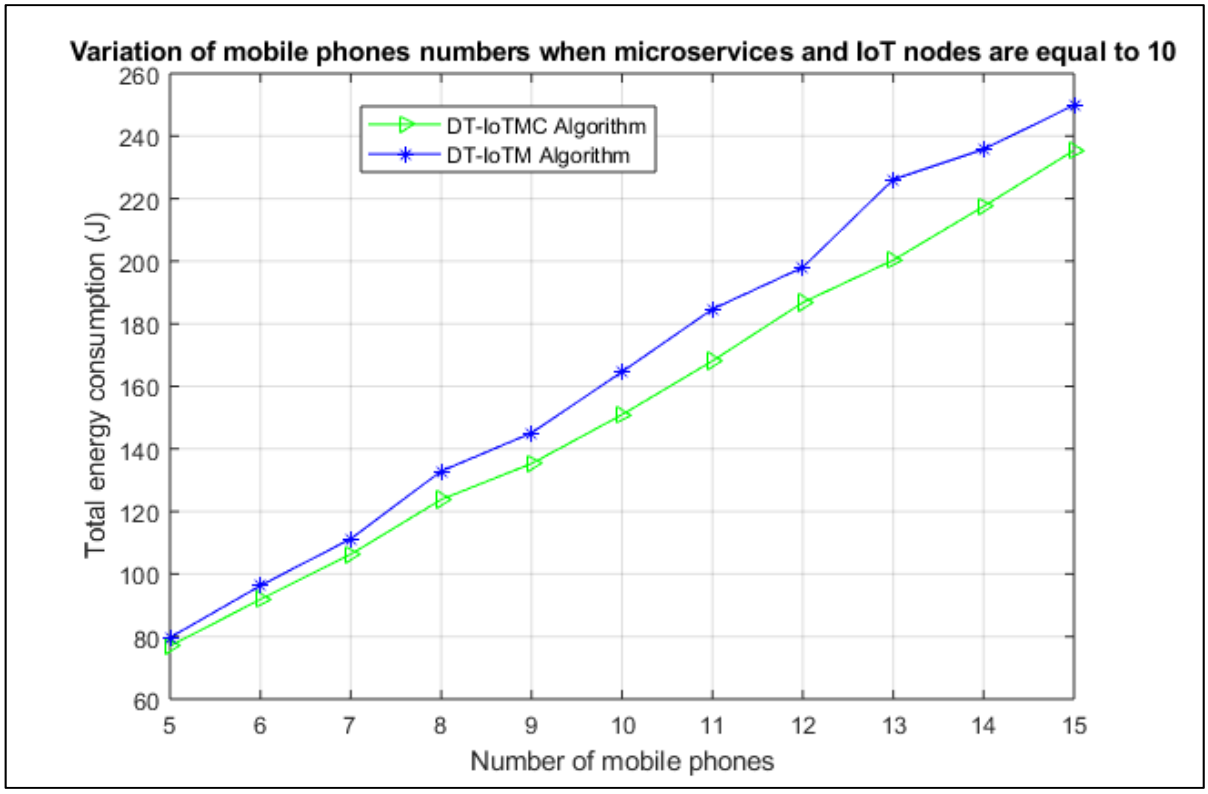

Figure 4-22 L'effet de la variation de nombre des appareils mobiles M sur la consommation totale d'énergie quand les microservices et le nombre N sont égales à 10

La figure 4-22 prouve que le deuxième algorithme donne la plus grande consommation totale d'énergie. Tandis que l'algorithme 1 donne les meilleurs résultats.

4.4.1.4 Comparaison des deux algorithmes avec la solution antérieure MUMTO-C

Dans cette partie, on va comparer les résultats obtenus avec une solution antérieure appelée MUMTO-C. cette solution est proposée récemment (Chen et al., 2018).
On va comparer nos algorithmes avec la solution MUMTO-C (barre rouge) qui effectue un déchargement à trois niveaux. On a fixé le nombre des nœuds IdO et des appareils mobiles à 5 et on va varier le nombre des tâches à décharger pour chaque nœud. Les simulations sont répétées 100 fois.

La figure 4-23 suivante présente les résultats obtenus:

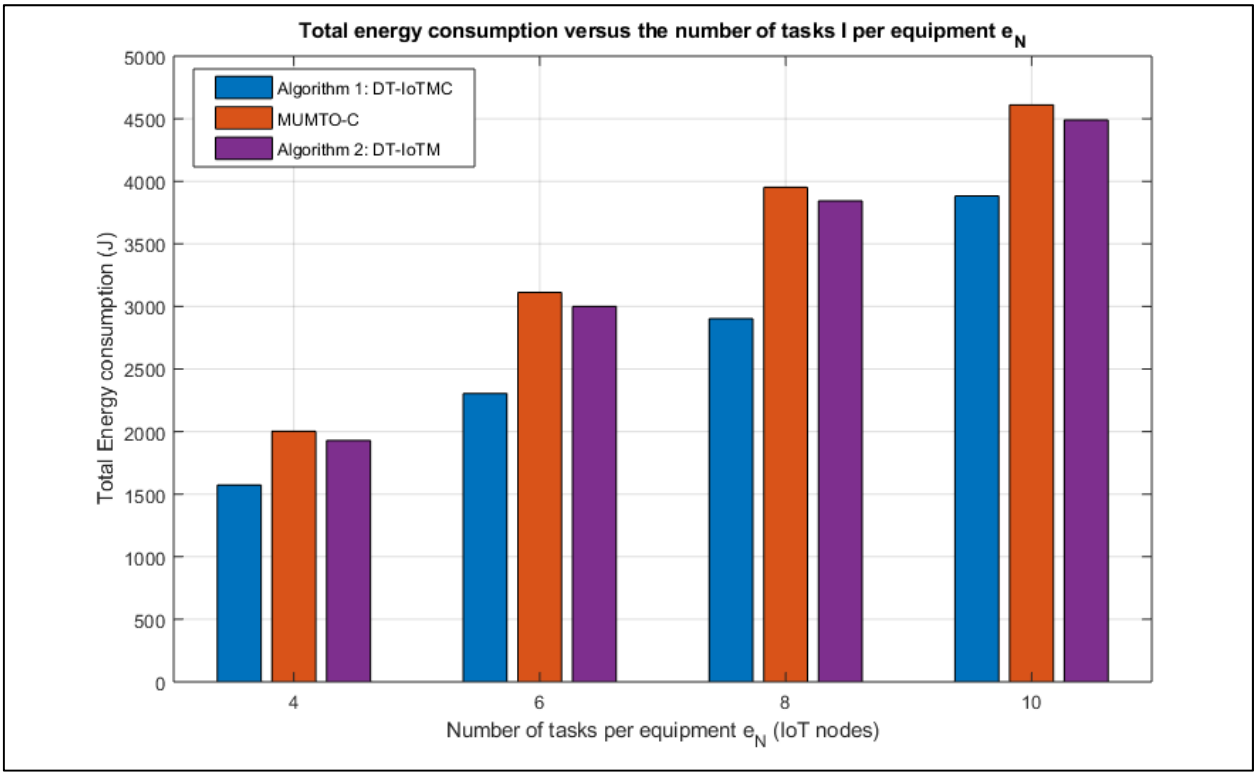

Figure 4-23 L'effet de la variation de nombre de tâches déchargeables par nœud IdO sur la consommation totale d'énergie en utilisant les deux algorithmes proposés et la solution Baseline

Nos algorithmes proposés (algorithme 1 et 2) donnent des bons résultats par rapport à la solution Baseline MUMTO-C (barre orangée). Cela s'explique par le fait que la solution antérieure traite un système avec des tâches et équipements multiples, mais une infonuagique distante et un point d'accès intermédiaire unique. Ainsi que le déchargement réalisé est à trois niveaux. Alors que nos solutions traitent un système de multi équipements IdO, multi-tâches, multi appareils mobiles et multi conteneurs sur l'infonuagique. Ce qui explique les bons résultats obtenus.

L'algorithme 1 (barre bleue) qui effectue un déchargement hybride entre des nœuds IdO multiples, des conteneurs multiples et des appareils mobiles multiples, donne les meilleurs résultats.

L'algorithme 2 (barre violette) qui suit un déchargement à deux niveaux vers les appareils mobiles et n'inclut pas les conteneurs sur l'infonuagique donne des résultats logiques mais mauvaises par rapport à l'algorithme DT_IoTMC.

L'algorithme DT_IoTMC a prouvé qu'il minimisait l'énergie totale du système par rapport à la solution Baseline avec une valeur située entre 15% et 27%.

Tandis que L'algorithme DT_IoTM a prouvé qu'il minimisait l'énergie totale du système par rapport à la solution Baseline avec une valeur située entre 2.6% et 3.7%.

En comparant les performances de nos deux algorithmes proposés, on a remarqué que DT_IoTMC minimisait l'énergie totale par rapport à DT_IoTM avec un pourcentage qui varie entre 14% et 24%.

4.4.2 Discussion

Dans toutes les simulations réalisées, on constate que notre premier algorithme DT-IoTMC, le déchargement hybride, donne la meilleure solution avec la moindre consommation totale d'énergie. Vu que la décision est prise parmi trois choix disponibles, comme le traitement local de côté nœuds IdO, le déchargement et le traitement distant du côté les appareils mobiles ou le déchargement et le traitement sur les conteneurs de côté infonuagique.

Par conséquent, le fait de décharger vers différents équipements et augmenter le spectre de choix aide à bien utiliser les ressources de différents éléments du système et à éviter leurs

surcharges. Spécialement si le déchargement va être basé sur l'état actuel de la bande passante entre les deux équipements communiquant et la fréquence de traitement de l'équipement. Comme exprimé dans le chapitre 3 (méthodologie), la consommation de l'énergie lors de déchargement sera affectée par l'énergie de communication et l'énergie de traitement.

Le deuxième algorithme DT_IoTM présente la possibilité de traiter les tâches localement ou de les décharger vers les appareils mobiles. Ce dernier donne la plus grande consommation totale d'énergie vue qu'il y a des tâches lourdes même en les traitant sur les appareils mobiles. Sans oublier la quantité de l'énergie consommée lors de déchargement vers un élément à distance, surtout si l'état actuel de la bande passante est médiocre.

L'absence des conteneurs sur l'infonuagique dans cette solution a marqué son importance, vu sa richesse en ressources et ses capacités abondantes de traitement.

Dans le scénario où on a fixé le nombre des nœuds IdO et des appareils mobiles à 5, et on a varié le nombre des tâches déchargeables. L'algorithme DT_IoTMC a réussi à améliorer la consommation énergétique totale de notre système par rapport à la solution Baseline par une valeur moyenne de 23%. Alors que l'algorithme DT_IoTM a donné une amélioration moyenne de 3% par rapport à la solution Baseline. En comparant nos deux algorithmes proposés, on a remarqué que l'algorithme DT_IoTMC a minimisé la consommation totale d'énergie du système par une valeur moyenne égale à 20% par rapport à l'algorithme DT_IoTM.

CONCLUSION GÉNÉRALE

Dans ce travail, nous abordons le problème de distribution optimisée des tâches dans un environnement hétérogène composé d'une plateforme équipée des nœuds multi-objet, une plateforme équipée des multi-appareils mobiles et une plateforme équipée des multi-conteneurs sur le nuage. Nous avons proposé une architecture distribuée basée sur des microservices implémentée dans les trois plateformes.

Nous avons introduit une formulation mathématique qui définit formellement notre problème comme un problème d'optimisation sous forme d'une programmation linéaire à nombres entiers mixte (MILP) avec variables binaires appelé DT_IoTMC « Distributed tasks between IoT nodes, Containers and mobile phones » et nous l'avons résolu avec le solveur intlinprog de MATALB. L'objectif de « DT_IoTMC » est de minimiser la consommation énergétique globale du système en tenant compte de l'état de réseau de communication, les limites des ressources dans chaque équipement, les capacités des batteries des nœuds IdO et des appareils mobiles tout en respectant les exigences des applications IdO en termes de délais d'échéances des tâches.

D'abord, nous avons conçu et implémenté des nœuds IdO pour une maison intelligente. Ensuite, nous avons développé une application mobile pour le contrôle de domicile. Nous avons mis en place une architecture des microservices implémentés sur les différents équipements du système et sur les conteneurs sur l'infonuagique, et un mécanisme de distribution des tâches entre les différents équipements du système. Ce dernier est composé de deux modules : un module de collecte des profileurs et un module d'optimisation. Le module de collecte des profileurs sert à collecter les informations nécessaires sur les équipements du système (nœuds IdO, appareils mobiles et conteneurs) comme les capacités en RAM, CPU et batteries ainsi que les bandes passantes entre les équipements communicants. Il déclenche le module d'optimisation lors de la détection d'un état critique du système. Le module d'optimisation exécute l'algorithme afin de distribuer les tâches et d'activer les microservices convenables sur l'équipement approprié.

Afin de résoudre le problème d'optimisation, nous avons développé deux algorithmes : DT_IoTMC qui suit un déchargement hybride et DT_IoTM qui suit un déchargement à deux niveaux.

Les résultats expérimentaux ont montré l'efficacité des algorithmes proposés dans les différents scénarios. On a évalué l'effet de variation de ces paramètres (nombre des nœuds IdO, nombre des tâches déchargeables et nombre des appareils mobiles) sur la consommation énergétique totale du système et la distribution des tâches. Dans le scénario de la variation du nombre des tâches à décharger et par rapport à la solution Baseline, on a constaté que l'amélioration de la consommation énergétique totale réalisée par nos algorithmes a donné les valeurs moyennes suivantes : 23% pour l'algorithme DT_IoTMC et 3% pour l'algorithme DT_IoTM.

Le fait de ne pas prendre en considération l'énergie consommée lors de l'exécution des scripts et de nos modules proposés (profiler collection module et optimization module) peut être considérer comme un point important à améliorer.

Dans le futur, nous pouvons élargir notre recherche pour étudier le problème de la file d'attente, où on peut calculer le nombre des tâches acceptées et rejetées dans un tel système afin d'éviter l'état de blocage. On peut aussi utiliser la plateforme Kubernetes qui permet l'automatisation de déploiement, la montée en charge et la mise en œuvre de conteneurs d'application sur des clusters de serveurs.

Nous avons rédigé un article de journal intitulé « Task optimization on microservices for smart home control applications » et l'avons soumis au journal « The IEEE Internet of Things Journal ».

BIBLIOGRAPHIE

Aazam, Mohammad & Zeadally, Sherali & Harras, Khaled. Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities. Future Generation Computer Systems, 87, 278-289.

Abd Rahim, M. R., Rashid, R. A., Rateb, A. M., Sarijari, M. A., Abdullah, A. S., Hamid, A. H. F. A., . . . Fisal, N. (2018). Service-Oriented Architecture for IoT Home Area Networking in 5 G. 5G Networks: Fundamental Requirements, Enabling Technologies, and Operations Management, 577-602.

Ahn, S., Lee, J., Park, S., Newaz, S. S., & Choi, J. K. (2017). Competitive partial computation offloading for maximizing energy efficiency in mobile cloud computing. IEEE Access, 6, 899-912.

Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. Computer networks, 54(15), 2787-2805.

Barbon, G., Margolis, M., Palumbo, F., Raimondi, F., & Weldin, N. (2016). Taking Arduino to the Internet of Things: the ASIP programming model. Computer Communications, 89, 128-140.

B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in Proc. 6th Eur. Conf. Comput. Syst., 2011, pp. 301–314.

Capgemini. (2016, January, 22). Microservices and its cousins - a quick summary of key 'service 'concepts. Repéré à https://www.capgemini.com/2016/01/microservices-its-cousins-a-quick-summary-of-key-service-concepts/

Carvalho, O., Garcia, M., Roloff, E., Carreño, E. D., & Navaux, P. O. (2017). IoT workload distribution impact between edge and cloud computing in a smart grid application. Dans Latin American High-Performance Computing Conference (pp. 203-217). Springer.

Chen, M.-H., Liang, B., & Dong, M. (2018). Multi-user multi-task offloading and resource allocation in mobile cloud systems. IEEE Transactions on Wireless Communications, 17(10), 6790-6805.

Chen, X. (2014). Decentralized computation offloading game for mobile cloud computing. IEEE Transactions on Parallel and Distributed Systems, 26(4), 974-983.

Chen, X., & Zhang, J. (2017). When D2D meets cloud: Hybrid mobile task offloadings in fog computing. Dans 2017 IEEE international conference on communications (ICC) (pp. 1-6). IEEE.

Dzone. (2016, June). Building microservices: Using an API Gateway. Repéré à https://dzone.com//articles/building-microservices-using

DZone. (2017, May, 17). What are microservices actually? Repéré à https://dzonw.com/articles/what/are/microservices-actually

Flores, H., Sharma, R., Ferreira, D., Kostakos, V., Manner, J., Tarkoma, S., Li, Y. (2017). Social-aware hybrid mobile offloading. Pervasive and Mobile Computing, 36, 25-43.

Le MagIT. (2014, décembre, 31). SOA et développement : à quoi faut-il s'attendre en 2015. Repéré à https://www.lemagit.fr/actualites/2240237456/SOA-et-developpement-a-quoi-faut-il-sattendre-en-2015

Martinfowler (2016, March, 1). Infrastructure as code, Repéré à https://martinfowler.com/bliki/InfrastructureAsCode.html

Namiot, D., & Sneps-Sneppe, M. (2014). On microservices architecture. International Journal of Open Information Technologies, 2(9), 24-27.

Objectcomputing. (2019). Case study: From monolith to microservices with MICRONAUT. Repéré à https://objectcomputing.com/about/our-clients/case-studies/case-study-from-monolith-to-microservices-with-micronaut

P. Kuendee and U. Janjarassuk (2018), "A comparative study of mixed-integer linear programming and genetic algorithms for solving binary problems," 2018 5th International Conference on Industrial Engineering and Applications (ICIEA), pp. 284-288.

P. Shu et al., ‘‘eTime: Energy-efficient transmission between cloud and mobile devices,’’ in Proc. INFOCOM, 2013, pp. 195–199

Prondroiddev. (2019, April). Moving towards a micro-service mindset on Android. Repéré à https://proandroiddev.com/moving-towards-a-micro-service-mindset-on-android-910de7e4f0c2

R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden, “Wishbone: Profile-based partitioning for sensornet applications,” in Proc. 6th USENIX Symp. Netw. Syst. Des. Implementation, 2009, vol. 9, pp. 395–408.

RedHat. (2017). Que sont les microservices? Repéré à https://www.redhat.com/fr/topics/microservices/what-are-microservices

Samsung. (2017). Smart things Hub. Repéré à https://www.samsung.com/ca/smartthings/hub/?cid=ca_ppc_google_IoT_SmartThingsRetention_%2Bsmart%20%2Bthings%20%2Bsmart%20%2Bhub_20180112

Stojkoska, B. L. R., & Trivodaliev, K. V. (2017). A review of Internet of Things for smart home: Challenges and solutions. Journal of Cleaner Production, 140, 1454-1464.

Studylibfr (2015, septembre-novembre). Microservices : Conception d’Applications Hétérogènes Distribuées. Repéré à https://studylibfr.com/doc/4353004/microservices

T. Vresk and I. Čavrak, “Architecture of an interoperable IoT platform based on microservices,” in 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2016, pp. 1196–1201

Whitmore, A., Agarwal, A., & Da Xu, L. (2015). The Internet of Things—A survey of topics and trends. Information Systems Frontiers, 17(2), 261-274.

Wu, H. (2018). Multi-objective decision-making for mobile cloud offloading: A survey. IEEE Access, 6, 3962-3976.

X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” IEEE/ACM Trans. Netw., vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

Y. Xiao and M. Krunz, “QoE and power efficiency tradeoff for fog computing networks with fog node cooperation,” in Proc. IEEE INFOCOM, Atlanta, GA, USA, May 2017, pp. 1–9.

Zhang, W., & Wen, Y. (2015). Energy-efficient task execution for application as a general topology in mobile cloud computing. IEEE Transactions on cloud Computing, 6(3), 708-719.

Zhang, W., Wen, Y., & Wu, D. O. (2014). Collaborative task execution in mobile cloud computing under a stochastic wireless channel. IEEE Transactions on Wireless Communications, 14(1), 81-93.