Placement des données de l'internet des objets dans une infrastructure de fog

Mohammed Islam Naas

THESE DE DOCTORAT DE

L'UNIVERSITE
DE BRETAGNE OCCIDENTALE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par
« Mohammed Islam NAAS »

« Placement des données de l'Internet des Objets dans une infrastructure de Fog »

Thèse présentée et soutenue à l'Université de Bretagne Occidentale, le 19/02/2019
Unité de recherche : UMR 6285

Rapporteurs avant soutenance :

Nathalie MITTON	Directrice de recherche, INRIA, Lille-Nord.
Sébastien MONNET	Professeur, Université de Savoie Mont Blanc, Chambéry.

Composition du Jury :

Président :	Olivier BEAUMONT	Directeur de recherche, INRIA, Bordeaux Sud-Ouest.
Examinateurs :	Nathalie MITTON	Directrice de recherche, INRIA, Lille-Nord.
	Sébastien MONNET	Professeur, Université de Savoie Mont Blanc, Chambéry.
	Philippe RAIPIN	Ingénieur de recherche, Orange Labs, Cesson-Sévigné.
	Laurent LEMARCHAND	Maitre de conférences, Université de Bretagne Occidentale, Brest.
Dir. de thèse :	Jalil BOUKHOBZA	Maitre de conférences, HDR, Université de Bretagne Occidentale, Brest.

Invité
Michel HURFIN — Chargé de recherche, HDR, INRIA, Rennes.

REMERCIEMENTS

Tout d'abord, je remercie **Allah**, **Le tout –miséricordieux**, **Le très –miséricordieux**, de m'avoir guidé à achever ce travail.

Je remercie chaleureusement l'équipe qui m'a encadrée sans laquelle cette thèse n'aurait jamais vu le jour. En premier lieu mon directeur de thèse, **Jalil Boukhobza**, celui que je voudrais exprimer ma reconnaisse pour sa confiance en moi, pour son travail de suivi, pour ses précieux conseils. Je tiens également à remercier **Philippe Raipin** pour avoir proposé le sujet de thèse, pour sa confiance en moi, pour son travail de suivi et pour ses remarques enrichissantes. Je tiens enfin à remercier **Laurent Lemarchand** pour son travail de suivi et pour ses contributions, tout en soulignant sa gentillesse et sa bienveillance.

Je remercie aussi les membres du jury : les professeurs **Nathalie Mitton**, **Sébastien Monnet** et **Olivier Beaumont** d'avoir accepté d'être rapporteurs et examinateurs de cette thèse.

Je remercie l'ensemble des membres de l'équipe DDSD à Orange Labs du site de Cesson-sévigné, son avoir oublié **Jérôme Bidet** et **Anne-Claire Bailleaux**, pour leur soutien et leur encouragement durant cette thèse.

Je remercie **Hamza Ouarnoughi**, **Djilali Boukhlef**, **Arezki Laga** et tous mes collègues du laboratoire Lab-STICC de l'UBO.

Je remercie mes parents, ma conjointe, tous les membres de ma famille et tous mes amis.

Enfin, j'adresse mes remerciements toute personne qui m'a aidée pour réaliser ce travail.

Mohammed Islam NAAS

PUBLICATIONS

Conférences internationales

1. M. I. Naas, J. Boukhobza, P. Raipin, and L. Lemarchand., An Extension to iFogSim to Enable the Design of Data Placement Strategies., In 2nd International Conference on Fog and Edge Computing (ICFEC), Washington DC, USA. May 2018. 8 pages.
2. M. I. Naas, L. Lemarchand, J. Boukhobza, and P. Raipin., A Graph Partitioning-based Heuristic for Runtime IoT Data Placement Strategies in a Fog infrastructure., In Symposium on Applied Computing (SAC), Pau, France. April 2018. 8 pages.
3. M. I. Naas, P. Raipin, J. Boukhobza, and L. Lemarchand., iFogStor : An IoT data placement strategy for fog infrastructure., In 1st International Conference on Fog and Edge Computing (ICFEC), Madrid, Spain. May 2017. 8 pages.

Communications

4. M. I. Naas, J. Boukhobza, P. Raipin, and L. Lemarchand., iFogStor : An IoT data placement strategy for fog infrastructure., In Meeting : Placement Problem in Context of Fog and Edge computing, Presentation. Grenoble, France. June 2018.
5. M. I. Naas, J. Boukhobza, P. Raipin, and L. Lemarchand., iFogStor : An IoT data placement strategy for fog infrastructure., In Performance and Scalability Storage Systems Workshop (Per3S), Poster and Presentation. Rennes, France. January 2018.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

ACRONYMES

AG Algorithme Génétique

CDN Content Delivery Network

CPU Central Processing Unit

GAP Generalized Assignment Problem

GPS Global Positioning System

GSM Global System for Mobile Communications

GNU GNU is Not UNIX

IoT Internet of Things

IDC International Data Corporation

IoV Internet of Vehicules

IPv6 Internet Protocol version 6

ILP Integer Linear Programming

IDE Integrated Development Environment

LPOP Local Point of Presence

MILP Mixed Integer Linear Programming

MEC Mobile Edge Computing

NFC Near Field Communication

NFV Network Functions Virtualization

OC Optimisation Combinatoire

OPL Optimization Programming Language

POP Point of Presence

POC Problème d'Optimisation Combinatoire

PL Programmation Linéaire

QoS Quality of Service

RFID Radio Frequency IDentification

RPOP Regional Point of Presence

RAM Random Access Memory

SDN Software-Defined Network

SLA Service Level Agreement

UML Unified Modeling Language

Wi-Fi Wireless Fidelity

6LowPAN IPv6 Low Power wireless Area Networks

Chapitre 1

INTRODUCTION

Sommaire

1.1 CONTEXTE GÉNÉRAL

L'Internet des objets (*Internet of Things*, IoT) est l'infrastructure qui sert à connecter tout objet à l'Internet, tels que des capteurs, des dispositifs portables (*Wearables*), des smartphones, des appareils photos, des appareils électroménagers et des véhicules [84, 97]. Au cours des dernières années, le nombre d'objets connectés n'a cessé d'augmenter. D'ici 2020, Cisco prédit que le nombre d'objets connectés dépassera 50 milliards [57]. Ce grand nombre d'objets générera une énorme quantité de données. Selon les estimations de l'International Data Corporation (IDC), 27% des données mondiales seront produites par des appareils mobiles et connectés [92].

Actuellement, les données de l'IoT sont généralement traitées et stockées dans le *Cloud* [62, 102]. Ce dernier satisfait la majorité des besoins des applications de l'IoT en termes d'accès ubiquitaire, de disponibilité et d'évolutivité des performances de traitement et de capacité de stockage.

Cependant, le *Cloud* étant un paradigme centralisé dans des centres de données, les données produites et les requêtes émises par les objets connectés sont transmises à ces centres situés dans le cœur de réseau. Ainsi, avec le nombre croissant d'objets connectés et la quantité de données produites, l'envoi de toutes les données vers le *Cloud* générera des goulets d'étranglement [62]. Ceci augmente la latence de transmission de données et, par conséquent, dégrade la

qualité de service (QoS) [83]. De nombreuses applications de l'IoT telles que l'Internet des Véhicules (IoV), la télémédecine et l'industrie 4.0 sont très sensibles aux latences. Le moindre retard dans les réponses du système peut entraîner des problèmes critiques [62].

Du fait des problèmes de latence et de trafic réseau générés par le *Cloud*, le paradigme de *Fog computing* a émergé [31]. Le Fog permet de décentraliser les services traditionnellement fournis par le *Cloud* sur l'ensemble des équipements réseaux. Il permet d'utiliser les ressources de calcul et de stockage des équipements réseaux tels que les routeurs et les switchs localisés entre les objets connectés et les centres de données du *Cloud* [51]. Le *Fog* permet le filtrage, l'agrégation et le traitement des données dans les périphéries du réseau, offrant une réduction de la latence, la préservation de la bande passante du réseau et l'amélioration de la qualité de service (QoS) [34].

Le *Fog* présente une infrastructure informatique dense et géo-distribuée à grande échelle. Les composants du *Fog*, également appelés nœuds de *Fog*, sont hétérogènes en termes de performances de traitement, de capacité de stockage et de latence d'accès pour les objets et les utilisateurs. Les nœuds ayant des ressources limités (ex. les set-top-boxes) sont plutôt situés en périphérie du réseau et fournissent une latence d'accès faible (de quelques millisecondes), tandis que les nœuds riches en ressources (ex. les points de présence) sont situés au cœur du réseau et fournissent un temps de latence d'accès plus élevée (de plusieurs secondes) [76]. Comme la localisation de données est cruciale pour les traitements, il est important d'étudier ***comment les données doivent être placées dans une telle infrastructure afin de réduire les latences dans le système*** [76, 99]. Par exemple, dans le cas ou un service agrège plusieurs données, leur traitement ne peut commencer qu'après la récupération de toutes les données requises. Un mauvais placement de données retarde donc le service.

De nombreux travaux de recherche ont étudié la minimisation de la latence du système dans le contexte du Fog et de l'IoT [25, 88, 90, 95]. Ces travaux ont essentiellement investigué le placement de services pour réduire la latence du système. Différentes solutions ont été proposées : des algorithmes exacts [60, 87], des heuristiques [95] ou des méta-heuristiques [72]. En revanche, le placement de services peut être contraint par d'autres propriétés que la latence, comme la sécurité (ex. les services peuvent être placés que dans des nœuds de *Fog* spécifiques), la consommation énergétique ou l'équilibrage de charge [46, 101]. Ceci rend la résolution du problème de placement de services difficile. Dans ce travail de thèse, nous avons étudié une autre méthode pour minimiser la latence du système : l'optimisation du placement des données d'IoT dans le *Fog*.

1.2 CONTEXTE INDUSTRIEL DE LA THÈSE

Le travail présenté dans cette thèse s'inscrit dans le cadre d'un contrat *CIFRE* entre l'entreprise Orange et l'Université de Bretagne Occidentale (UBO). La thèse s'est déroulée essentiellement dans les locaux d'Orange Labs à Cessons-sévigné, une filiale dans laquelle Orange mène des travaux de recherche. Le travail de la thèse est effectué dans l'équipe DDSD (Distibuted Data System Design) dont la thématique est centrée sur la gestion de stockage de données dans le *Cloud*. La thèse s'est déroulée à l'UBO dans l'équipe MOCS (Methods, tOols, Circuits/-Systems) du laboratoire Lab-STICC [8]. La thématique de MOCS porte sur les méthodologies et les outils de conception des matériels/ logiciels. Ce travail de thèse s'inscrit en particulier dans le thème de l'optimisation du stockage de données.

Orange est une entreprise leader dans le domaine des télécommunications. En effet, elle compte plus que 270 millions de clients répartis dans 29 pays [12]. Avec l'avènement de l'IoT, Orange a investie largement dans les services de cette technologie. Actuellement, orange gère au quotidien plus de 16 millions d'objets connectés et traite plus de 375 millions de données chaque minute [13]. Elle contribue aussi à lever les verrous scientifiques existants dans l'IoT et dans ses plate-formes de déploiement, notamment le *Fog* et le *Cloud*. En effet, elle a crée le projet *Fog computing* qui a pour but d'optimiser l'utilisation des ressources dans les infrastructures de *Fog*. Cette thèse, comme plusieurs autres, s'inscrit dans ce projet.

1.3 PROBLÉMATIQUES

Notre travail consiste à investiguer une méthode de placement de données de l'IoT dans les infrastructures *Fog* pour minimiser la latence du système. Cette problématique donne lieu à plusieurs sous problèmes décrits ci-dessus. Dans cette thèse, nous proposons une solution pour chaque sous problème.

1.3.1 *Minimisation de la latence du système dans les infrastructures de Fog*

Comme mentionné précédemment, le *Fog computing* présente une infrastructure hétérogène et géo-distribuée. Le mauvais placement des données dans une telle infrastructure peut entraîner une augmentation de la latence globale du système. Ainsi, la première problématique abordée dans cette thèse est : comment les données d'IoT doivent elles être placées dans une infrastructure de *Fog* afin de minimiser la latence globale de système ?

1.3.2 *Réduction du temps de placement de données d'IoT dans le Fog*

Les infrastructures du *Fog* sont dynamiques et leur topologie de réseau change fréquemment. Cela est dû, par exemple, à la mobilité des objets connectés ou aux migrations de services d'IoT. Par conséquent, les stratégies de placement et de migration de données doivent être conçues pour être exécutées périodiquement et en ligne. Cependant, rechercher la meilleure stratégie de placement pour de telles infrastructures de grande taille (des millions d'objets et des milliers de nœuds de *Fog* et de services d'IoT) peut entraîner une explosion combinatoire en termes de temps de calcul et d'espace mémoire. Cela rend une telle stratégie de placement de données irréalisable pour un usage en ligne.

Une façon d'accélérer le temps de calcul du placement de données dans le *Fog* consiste à utiliser des heuristiques basées sur des approches de type "diviser pour régner". Dans ces approches, le problème de placement d'origine est subdivisé en plusieurs sous-problèmes résolus indépendamment les uns des autres. Chaque sous-problème est défini par ses propres ensembles de noeuds de Fog, de données et de services. La taille des sous-problèmes traités étant moins importante, leur temps de résolution est réduit, ce qui permet une résolution plus rapide du problème global. Cependant, augmenter le nombre de sous-problèmes pour accélérer les calculs induit une perte d'information sur l'espace de recherche de la solution. Plus précisément, en décomposant le problème de placement en plusieurs sous-problèmes, certaines données et leurs consommateurs seront affectés à des sous-problèmes différents. Cela dégrade la qualité de la solution de placement trouvée. La deuxième problématique adressée dans cette thèse consiste donc à étudier comment réduire le temps de calcul du placement de données dans le contexte du *Fog* et de l'IoT.

1.3.3 *Évaluation des stratégies de placement de données d'IoT dans le Fog*

Comme mentionné auparavant, les infrastructures de *Fog* peuvent inclure des milliers de nœuds et d'instances de service d'IoT et desservir des millions d'objets connectés. Les stratégies de placement de données pour de tels systèmes complexes doivent être explorées et évaluées à l'aide de méthodes de simulation [62]. Ceci permet de réduire les coûts et les délais d'évaluation, avant de déployer ces solutions dans des systèmes réels. Les simulateurs existants ne permettent pas la gestion du placement de données dans le contexte du *Fog* et de l'IoT. Il est donc nécessaire de proposer une extension à un simulateur existant afin de le rendre compatible avec nos besoins. La troisième problématique adressée dans cette thèse est donc : quel simulateur existant pouvons-nous étendre et comment pour tester et évaluer nos stratégies de placement de données de l'IoT dans les infrastructures de *Fog* ?

1.4 CONTRIBUTIONS

Dans cette partie, nous décrivons les différentes contributions de ce travail de thèse qui répondent aux problématiques soulevées dans la section précédente.

1.4.1 *Formulation du problème de placement de données dans le Fog*

Ce travail répond à la première problématique de cette thèse. Nous avons modélisé le problème de placement de données comme un problème d'affectation généralisé (Generalized Assignment Problem, GAP) [29]. GAP consiste à trouver une affectation qui minimise le coût de placement d'un ensemble de tâches dans un ensemble d'agents, sachant que chaque agent a une capacité de traitement, et chaque tâche a une taille et un coût différents par rapport à l'agent choisi. GAP adresse parfaitement notre problème. Plus précisément, les tâches modélisent les données, les agents modélisent les noeuds de Fog et le coût de l'affection représente la latence du système. Dans cette modélisation, nous avons utilisé des méthodes basées sur la programmation linéaire en nombres mixtes. En effet, il existe des outils efficaces pour résoudre ce genre de problèmes. Par exemple, *CPLEX* peut résoudre des problèmes similaires avec des millions de variables et des dizaines de milliers de contraintes [6].

1.4.2 *Stratégies de placement proposées*

Après avoir formulé le problème de placement de données d'IoT dans le *Fog* comme un programme linéaire, nous avons proposé trois méthodes pour le résoudre : une approche exacte et deux heuristiques. Ces méthodes sont décrites ci-après :

1.4.2.1 *Approche exacte : iFogStor*

Cette solution consiste à résoudre le problème du placement de données comme un seul programme linéaire en utilisant le solveur CPLEX MILP [6]. *iFogStor* trouve une solution optimale pour stocker les données en minimisant la latence globale du système.

1.4.2.2 *Heuristique 1, partitionnement géographique de l'infrastructure : iFogStorZ*

Au vu de la complexité exponentielle du problème de placement, nous avons proposé *iFogStorZ*, une heuristique basée sur le concept de "diviser pour régner". *iFogStorZ* subdivise l'infrastructure du problème en plusieurs zones géographiques. Pour chacune des zones, un sous-problème de placement avec une complexité moindre est modélisé et résolu en utilisant l'approche exacte *iFogStor*.

La solution globale de placement est construite par l'agrégation des solutions de tous les sous-problèmes résolus.

Les travaux d'*iFogStorZ*, d'*iFogStor* et de la formulation du problème de placement ont été publiés dans la conférence "The IEEE International Conference on Fog and Edge Computing", Madrid, Espagne, mai 2017 [76].

1.4.2.3 *Heuristique 2, partitionnement de l'infrastructure à base de la théorie des graphes : iFogStorG*

Comme nous l'avons expliqué avant, la subdivision géographique d'infrastructures peut induire une perte d'information sur l'espace de recherche de la solution, et par conséquent, un mauvais placement de données. Pour éviter cela, nous avons proposé *iFogStorG*, une heuristique qui subdivise l'infrastructure en utilisant des méthodes basées sur la théorie des graphes. Dans cette heuristique, l'infrastructure du système est modélisée dans un premier temps sous-forme d'un graphe pondéré en noeuds et en arêtes. Une fois le graphe pondéré, il est partitionné en plusieurs sous-graphes disjoints dont chacun représente un nouveau (sous-) problème de placement de donnée avec une complexité moindre. Comme dans *iFogStorZ*, les sous-problèmes de placement sont modélisés et résolus d'une manière indépendante, puis les solutions sont agrégées pour construire la solution globale de placement.

Ce travail a été publié dans la conférence "The 33rd ACM/SIGAPP Symposium on Applied Computing, SAC", Pau, France, avril 2018 [78].

1.4.3 *Extension du simulateur iFogSim*

Cette contribution répond à la troisième problématique de cette thèse. Nous avons proposé une extension au simulateur *iFogSim* [62]. L'objectif de cette extension est de permettre aux utilisateurs de déployer, de tester et d'évaluer leurs stratégies de placement de données d'IoT dans les infrastructures de *Fog* avant de les déployer sur des plate-formes réelles[1].

Ce travail a été publié dans la conférence "The IEEE International Conference on Fog and Edge Computing", Washington DC, États-Unis, mai 2018 [77].

1.5 PLAN DU MÉMOIRE

Ce manuscrit de thèse est composé de sept chapitres organisés en trois parties. La première partie regroupe le contexte de notre étude et un état de l'art. La

1 Le code source de cette extension est disponible à :
https ://github.com/medislam/iFogSimWithDataPlacement.git.

deuxième partie décrit chacune de nos contributions. La troisième partie conclut ce document.

Première partie : contexte et état de l'art

Le chapitre 2 définit le contexte et résume les concepts généraux de l'IoT, du *Fog* et de l'optimisation combinatoire utilisés dans nos contributions. Le modèle de notre système ainsi que les hypothèses utilisés de ce travail de thèse sont également donnés dans ce chapitre.
Le chapitre 3 présente un état de l'art en lien avec l'ensemble de nos contributions.

Deuxième partie : contributions

Le chapitre 4 décrit notre première contribution sur l'optimisation du placement de données de l'IoT dans le *Fog*. Nous donnons tout d'abord un aperçu de notre stratégie de placement. Ensuite, nous formulons le problème de placement comme un programme linéaire. Puis, nous proposons une méthode exacte (*iFogStor*) et une heuristique (*iFogStorZ*) pour le résoudre. Enfin, nous évaluons les solutions de placement proposées en utilisant un scénario générique de "ville intelligente".
Le chapitre 5 est consacré à l'heuristique de placement *iFogStorG*. Nous commençons par donner un aperçu de cette heuristique. Ensuite, nous montrons les différentes étapes de sa conception. Nous y décrivons enfin la méthode et les résultats d'évaluation de cette heuristique.
Le chapitre 6 présente notre extension du simulateur *iFogSim* afin d'y ajouter la gestion du placement de données. Nous commençons par présenter ce simulateur. Ensuite, nous décrivons les différents composants ajoutés à *iFogSim* pour l'objectif susmentionné. Enfin, nous évaluons la surcharge ajoutée à *iFogSim* par notre extension en termes de temps de simulation et d'utilisation mémoire.

Troisième partie : conclusion

Le chapitre 7 conclut le manuscrit en résumant les différentes contributions et en présentant quelques perspectives.

Première partie

CONTEXTE ET ÉTAT DE L'ART

Chapitre 2

CONTEXTE

L'IoT et le *Fog computing* constituent les domaines cibles de ce travail. Pour cette raison, dans ce chapitre, nous présentons les différents concepts relatifs à ces deux domaines. De plus, au vu de la nature du problème traité dans cette thèse, nous donnons également quelques notions de base sur l'optimisation combinatoire. Nous finissons ce chapitre par présenter le modèle d'architecture considéré ainsi que les hypothèses de ce travail.

Sommaire

2.1 DÉFINITIONS

Dans cette section, nous définissons un capteur, un actionneur et un objet connecté, termes relatifs à l'IoT.

Définition 2.1. ***Capteur*** *[56] : Les capteurs sont des dispositifs électroniques composés de cellules sensorielles capables de mesurer des paramètres physiques de l'environnement tels que la luminosité, la température, les sons, les mouvements ou tout autre paramètre de l'environnement. En général, les capteurs sont associés à des passerelles pour transmettre les données capturées à la plate-forme de traitement et de stockage.*

Définition 2.2. ***Actionneur*** *[84] : un actionneur est un dispositif électronique et/ou mécanique qui produit des actions pour modifier l'état ou le comportement d'un système.*

Définition 2.3. ***Objet connectés*** *[56, 68] : ou «objet intelligent» est tout objet ou produit qui, par le biais de technologies intégrées, est capable d'être identifié, de collecter des données sur l'état physique de son environnement, de prendre des décisions et de produire des actions (mécaniques ou par messages de contrôle), et de se connecter à un réseau (ex. Internet) pour échanger des données.*

2.2 INTERNET DES OBJETS

L'IoT est une infrastructure de réseau dynamique et globale dans laquelle les objets physiques (ex. accessoires, vêtements, machines, véhicules, etc.) sont autonomes et auto-configurables [22]. Ces objets sont censés être capables, grâce à l'utilisation des technologies de l'information et de télécommunication, d'envoyer et de recevoir des données sur Internet, de surveiller leur environnement et d'effectuer des actions basées sur le partage d'information [24]. Dans ce contexte, la domotique, la logistique, la télémédecine et l'apprentissage en ligne sont quelques exemples de scénarios d'application possibles dans lesquels l'IoT joue un rôle important.

Le terme «IoT» a été utilisé pour la première fois par Kevin Ashton en 1999 [65]. Les étiquettes d'identification par radiofréquence (RFID), les réseaux de capteurs, l'Internet, l'informatique intelligente sont considérés comme des technologies clés de l'IoT [65]. Après une décennie d'évolution, l'IoT est considéré aujourd'hui comme l'une des technologies révolutionnaires de ce siècle [64]. En effet, cette technologie est considérée comme un moyen pour améliorer les activités quotidiennes, la création de nouveaux modèles économiques de gestion de produits et de services [64].

l'IoT offre de grandes opportunités de marché pour les fabricants d'équipements, les fournisseurs de services Internet et les développeurs d'applications. Selon une analyse faite par Cisco, le nombre d'objets connectés dépassera les

50 milliards en 2020 offrant un marché mondial de 2.700 à 6.200 milliards de dollars annuellement [24, 57]. L'IoT présente aussi de bonnes opportunités pour l'emploi. En effet, d'ici 2020, 4,5 millions développeurs vont contribuer à des projets d'IoT dans le monde [10]. D'autre part, les secteurs de l'automobile et des transports verront plus de 220 millions de voitures connectées en 2020 [59]. Les applications de santé et d'industrie ont également un grand impact économique. Les applications de soins de santé et les services connexes basés sur l'IoT, tels que la santé mobile et la télémédecine, devraient générer une croissance d'environ 1.100 à 2.500 milliards de dollars chaque année d'ici 2025. La Figure 1 [24] montre l'impact économique potentiel des applications de l'IoT.

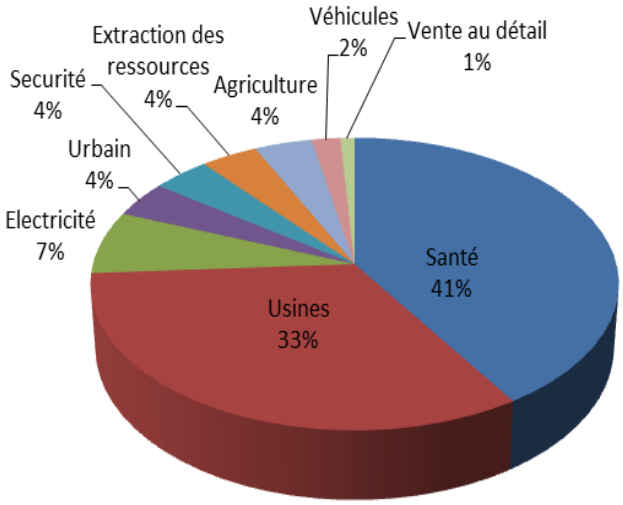


FIGURE 1 : L'impact économique potentiel des applications de l'IoT.

2.2.1 *Domaines d'applications de l'IoT*

Les domaines d'application de l'IoT sont classifiés par *Atzori et al.* dans [27] comme suit :

- **Transport et logistique :** Les voitures, les bus, les taxis ainsi que les trains sont de plus en plus équipés de capteurs, d'actionneurs et de puissance de traitement. Les routes elles-mêmes et les marchandises transportées sont également équipées par des étiquettes et des capteurs qui envoient des informations aux sites de contrôle et aux véhicules de transport pour mieux acheminer le trafic, aider à la gestion des dépôts, et fournir les marchandises transportées. L'une des applications réussies de l'IoT dans le secteur des transports est *Waze* [18], une application mobile de navigation GPS en temps réel. Cette application offre à ses utilisateurs des informations sur l'état du trafic en temps réel. Les utilisateurs, à leur tour, peuvent modifier la carte (sens de circulation, nouvelles rues, bouchons, etc.) afin de l'amé-

liorer ou de suivre l'évolution du réseau routier. Actuellement, le nombre d'utilisateurs de *Waze* dépasse les 100 millions dans le monde [19].

Une autre application de transport utilisant l'IoT est celle de *Skydrone* ; une entreprise française qui offre à ses clients la possibilité de livrer leurs objets par drones autonomes [17]. Les drones disposent d'une capacité de poids de charge (variable selon le modèle du drone) et ils sont équipés d'un système de détection d'obstacles aériens leur permettant d'être autonomes. Cette solution de transport aérien permet des livraisons rapides et flexibles.

- **Télémédecine :** La télémédecine est une forme de pratique médicale à distance utilisant des technologies de l'information et de la communication. Elle permet d'établir un diagnostic ou d'assurer un suivi pour un patient à distance. Par exemple, dans un système de santé à distance, la concentration quotidienne en oxygène, la glycémie et la pression artérielle d'un patient sont collectées automatiquement par des capteurs et ensuite transmises aux organismes de suivi. Une autre application de télémédecine consiste à offrir des soins rapides pour les personnes ayant subi un accident ou un problème de santé. Pour cette dernière, *Google* propose une solution à base de drones pour porter secours dans diverses situations d'urgence. Ces drones livrent un kit de premiers secours, des outils de diagnostic ou des médicaments [16].

- **Environnements intelligents :** Les maisons, les villes et les usines sont des exemples majeurs d'environnements intelligents. Les applications de ce domaine consistent à collecter et traiter des données de l'environnement pour réaliser des économies, sécuriser ou améliorer la vie des citoyens. Par exemple, dans le cas des maisons et des bureaux intelligents, le chauffage peut être adapté selon les conditions de la météo ; l'éclairage de la pièce peut être changé en fonction de l'heure de la journée ; les incidents peuvent être évités avec des systèmes de surveillance et d'alarme appropriés ; et l'énergie peut être économisée en éteignant automatiquement les équipements électriques non nécessaires [27]. Dans le cas de l'industrie, des usines peuvent être enrichies par des capteurs et des actionneurs pour contrôler la chaîne de production, trier les produits défaillants ou arrêter le processus de production.

- **Relation sociale :** Les applications relevant de ce domaine sont celles qui permettent à l'utilisateur d'interagir avec d'autres personnes afin de maintenir et de développer des relations sociales. En effet, les objets peuvent automatiquement déclencher la transmission de messages à des amis pour leur permettre de savoir ce que la personne fait ou ce qu'elle a fait dans le passé. Quelques applications dans ce domaine sont : de partager des avis sur des produits, des services ou des endroits [27].

- **Applications du futur :** Les applications décrites dans les sections précédentes sont déjà déployées. En dehors de celles-là, nous pouvons envisager de nombreuses autres applications futures. Parmi ces applications on peut noter : l'œil intelligent [11]. Cette technologie est très similaire aux lunettes de *Google* [7]. Elle consiste à équiper les lentilles de contact avec des capteurs et des capacités de connectivité comme le Wi-Fi et le Bluetooth pour fournir de nombreuses fonctions telles que parcourir des cartes, lire des emails ou des messages, naviguer sur Internet ou capturer des vidéos.

2.2.2 *Les défis de l'IoT*

De nombreux travaux ont cité les défis à relever pour assurer le bon déploiement des applications de l'IoT [24, 27, 61, 66, 81, 96]. Dans cette section, nous en décrivons quelques uns.

1. *Normalisation :* De nombreux fournisseurs et constructeurs proposent des services et des objets en utilisant leurs propres technologies, qui ne sont pas accessibles aux autres. Il est donc nécessaire de proposer des normes pour les objets, les protocoles de communication, les données et les architectures de déploiement pour assurer une meilleure interopérabilité entre différents objets et services. Des efforts considérables ont été déployés pour normaliser le paradigme de l'IoT par les communautés scientifiques comme les organismes européens de normalisation (ETSI, CEN, ENELEC, etc.) et les institutions de normalisation (ISO, UIT). Mais, la norme universelle reste néanmoins un défi pour l'IoT [66, 81, 96].

2. *Réseau et connectivité :* L'IoT consiste à connecter n'importe quels objets à n'importe quel moment. La connectivité est donc un ingrédient de base de cette technologie [81]. Dans l'IoT, la nécessité d'avoir un identifiant unique pour chaque objet exige la mise en place d'une politique d'adressage efficace. Comme nous savons qu'il y aura des milliards d'objets qui seront connectés dans le monde dans le futur proche, il existe un besoin dun système efficace pour la gestion des identités des objets. Actuellement, le protocole IPv6 peut adresser jusqu'à 1564 objets par mètre carré [3]. Par ailleurs, les objets doivent supporter plusieurs technologies de communication afin de fournir une bonne interopérabilité. Par exemple, la plupart des smartphones actuels supportent plusieurs technologies de communication comme le Wi-Fi, NFC, et GSM [24]. La mise au point des protocoles de communication et de routage efficaces en termes d'énergie, d'évolutivité et de performances, constitue actuellement un défi majeur de recherche pour l'IoT [24].

3. *Qualité de service :* La QoS constitue un autre défi pour le déploiement des applications de l'IoT. Parmi les éléments de la QoS qui doivent être améliorés, nous citons la disponibilité, la fiabilité, les performances, et l'interopérabilité [70].

    - la disponibilité dans un système d'IoT consiste à fournir des services aux clients n'importe où et à n'importe quel moment. Afin d'assurer la continuité des services et le support de toutes natures d'objets, les systèmes doivent supporter diverses plates-formes et protocoles, notamment celles de communication (ex. Wi-Fi, Bluetooth, Cellulaire, 6LowPAN, etc.). Par ailleurs, les services critiques doivent être fournis avec des solutions redondantes afin d'assurer leur disponibilité [24].
    - la fiabilité fait référence au fonctionnement du système par rapport aux attentes fixées dans les spécifications [24]. La fiabilité est critique et impose des exigences strictes dans le cas des systèmes critiques (ex. la gestion de trafic, l'industrie et la santé).
    - les performances des services représentent également un défi pour la QoS des applications de l'IoT. Plusieurs métriques sont utilisées pour évaluer les performances du système notamment la vitesse de traitement et le débit de transmission [24].
    - la gestion de l'interopérabilité sur des applications à grande échelle composées d'un grand nombre d'objets hétérogènes appartenant à différentes plates-formes représente un défi pour l'IoT [70]. L'interopérabilité doit être prise en compte par les développeurs d'applications et les fabricants d'objets afin de garantir la fourniture de services à tous les clients, quelles que soient les plates-formes matérielles et logicielles utilisées [24].

4. *Mobilité :* La mobilité de son coté représente un défi pour l'IoT car la plupart des services sont censés être fournis à ou par des objets mobiles. La mise en relation des utilisateurs mobiles avec les services demandés est un défi pour l'IoT. Une interruption de service pour les objets mobiles peut se produire lorsque ces objets sont transférés d'une passerelle à une autre [24]. Dans [55], les auteurs proposent un mécanisme de gestion de mobilité de ressources en deux modes : la mise en cache et la tunnelisation du réseau pour assurer la continuité du service. Ces méthodes permettent aux applications d'accéder aux données en cas d'indisponibilité temporaire des ressources. Le grand nombre d'objets connectés dans les systèmes de l'IoT nécessite également des mécanismes efficaces pour la gestion de la mobilité [24].

5. *Passage à l'échelle :* Le passage à l'échelle dans l'IoT signifie la possibilité d'ajouter de nouveaux objets, services et fonctions sans affecter la QoS. L'ajout de nouvelles opérations et la prise en charge de nouveaux objets est un défi dans le contexte de l'IoT, notamment en présence des plates-formes matérielles et des protocoles de communication hétérogènes. Les applications de l'IoT doivent être extensibles d'une manière transparente [24].

6. *Sécurité et confidentialité :* La sécurité et la confidentialité des données des utilisateurs représentent des défis importants pour l'IoT [24]. Cela résulte de plusieurs raisons. Premièrement, les composants de l'IoT sont souvent laissés sans surveillance, ce qui facilite les attaques physiques. Deuxièmement, la plupart des communications sont faites avec des réseaux sans fil, ce qui rend l'écoute très facile. Enfin, la plupart des composants de l'IoT se caractérisent par une faible capacité en termes d'énergie et de ressources de traitement, et par conséquent, ces composants ne peuvent pas mettre en œuvre des mécanismes complexes de sécurité [27]. Dans un environnement d'IoT, le système peut être attaqué de plusieurs façons : en désactivant la disponibilité du réseau, en injectant des données erronées dans le réseau ou en accédant à des informations personnelles [61].

Après avoir listé les défis majeurs de l'IoT, nous mentionnons ici que ce travail de thèse adresse les défis associés à la QoS. Plus précisément, il sert à améliorer le temps de transfert de données dans l'infrastructure du système.

Dans la section suivante, nous décrivons la solution actuelle de stockage et de traitement de données de l'IoT [62] ainsi que les problèmes présentés par cette solution.

2.2.3 *Le Cloud et l'IoT*

Actuellement, la majorité des applications de l'IoT sont déployées dans le *Cloud* [62]. Cette méthode satisfait le besoin de l'IoT en termes d'accès ubiquitaire, haute disponibilité et évolutivité de performances de calcul et de capacité de stockage. De plus, le *Cloud* libère les utilisateurs de la spécification de nombreux détails (ex. plates-formes, dépendances, etc.) et simplifie le déploiement et l'intégration des services [26, 31]. Cependant, le paradigme du *Cloud* est un modèle informatique centralisé. Cela signifie que toutes les données et les requêtes doivent être transmises aux centres de données du *Cloud* qui représentent des points centralisés dans le cœur du réseau [63]. Avec l'augmentation du nombre d'objets et la quantité de données produite, cette solution souffre de nombreux problèmes. Le transfert de données depuis les périphéries de réseau jusqu'aux centres de données peut causer des goulets d'étranglement, et des latences élevées et imprédictibles. Ainsi, la QoS en sera dégradée [83]. Par ailleurs, il existe de nombreuses

applications d'IoT qui sont sensibles à la latence, telles que la télémédecine, l'Internet des véhicules et la sécurité (ex. détection d'intrusion) [26]. De plus, certaines décisions peuvent être prises localement (i.e. à la périphérie du réseau), sans avoir à être transmises au *Cloud* [63]. D'autre part, comme mentionné dans la section 2.2.2, plusieurs applications d'IoT nécessitent un support de mobilité, un déploiement géo-distribué et une connaissance de la localisation que le *Cloud* ne peut pas assurer efficacement [99]. Ces problèmes ont permis l'émergence d'un nouveau paradigme, le *Fog computing*, détaillé dans la section suivante.

2.3 FOG COMPUTING

Le *Fog computing* ou "l'informatique en brouillard" est un paradigme proposé par Cisco en 2012 essentiellement pour faire face aux problématiques des latences élevées et du trafic réseau important causés par l'utilisation du *Cloud* [26, 71]. Il est considéré comme une extension du *Cloud* du cœur jusqu'aux périphérie du réseau (**F**rom c**O**re to ed**G**e, en Anglais) [63, 71, 83, 98]. Son idée principale consiste à utiliser les ressources disponibles dans des équipements du réseau localisés entre les centres de données du *Cloud* et les utilisateurs finaux (et éventuellement les objets connectés) pour faire une partie du traitement et du stockage de données avant de les envoyer au *Cloud*. Quelques exemples de ces équipements : les switchs, les routeurs, les passerelles et les stations de base. Le *Fog* permet le traitement, le filtrage et l'agrégation de données dans le cœur ainsi que dans les périphéries du réseau en permettant ainsi une réduction de la latence, la préservation de la bande passante réseau et l'amélioration de la QoS [33]. Contrairement au *Cloud*, le *Fog* a la capacité d'héberger des applications de l'IoT qui nécessitent des réponses rapides et en temps réel [21].

Le *Fog* ne remplace pas le *Cloud*, mais ces deux paradigmes de calcul sont complémentaires [80]. En effet, ils coopèrent pour fournir une plate-forme extensible qui supporte à la fois des applications nécessitant des latences courtes et prédictibles ainsi que des applications nécessitant des traitements complexes et du stockage de données permanent [83, 98]. L'idée est de servir les requêtes nécessitant des latences courtes (ex. contrôle du trafic routier, parking intelligent et diffusion en direct) par des équipements localisés dans le *Fog*, tandis que les requêtes nécessitant des traitements complexes et des données d'historisation (ex. photos, flux vidéos, données des réseaux sociaux, etc) sont servies par le *Cloud* [83].

Dans le *Fog computing*, les équipements de traitement et de stockage de données sont appelés les "*Nœuds du Fog*". Ces équipements sont de nature hétérogène au regard de leur performance [98]. Ils peuvent être des équipements modestes en ressources comme des set-top-boxes, des points d'accès, des switchs et des routeurs, ou des équipements riches en ressources comme les points de

présence[1] (POP). De plus, la distance géographique entre les nœuds du *Fog* et les utilisateurs finaux (et les objets connectés) est très variable ; de quelques mètres comme les points d'accès jusqu'à plusieurs centaines de kilomètres comme les POP. En effet, dans les infrastructures de type *Fog*, les équipements modestes en ressources sont localisés près des utilisateurs finaux et ils fournissent des latences courtes (qui se comptent en millisecondes), tandis que les équipements riches en ressources sont localisés loin des utilisateurs finaux et fournissent des latences importantes et imprédictibles (qui se comptent en secondes, voir en minutes) [42].

2.3.1 *Architecture du Fog*

Le *Fog* présente une infrastructure virtuelle et géo-distribuée intégrant un grand nombre d'équipements hétérogènes et qui sont interconnectés [45]. Comme illustré dans la Figure 2, l'infrastructure du *Fog* comprend les périphéries du réseau, le niveau d'agrégation et le cœur de réseau. Ci-après, nous décrivons chaque niveau de l'infrastructure illustrée dans la Figure 2.

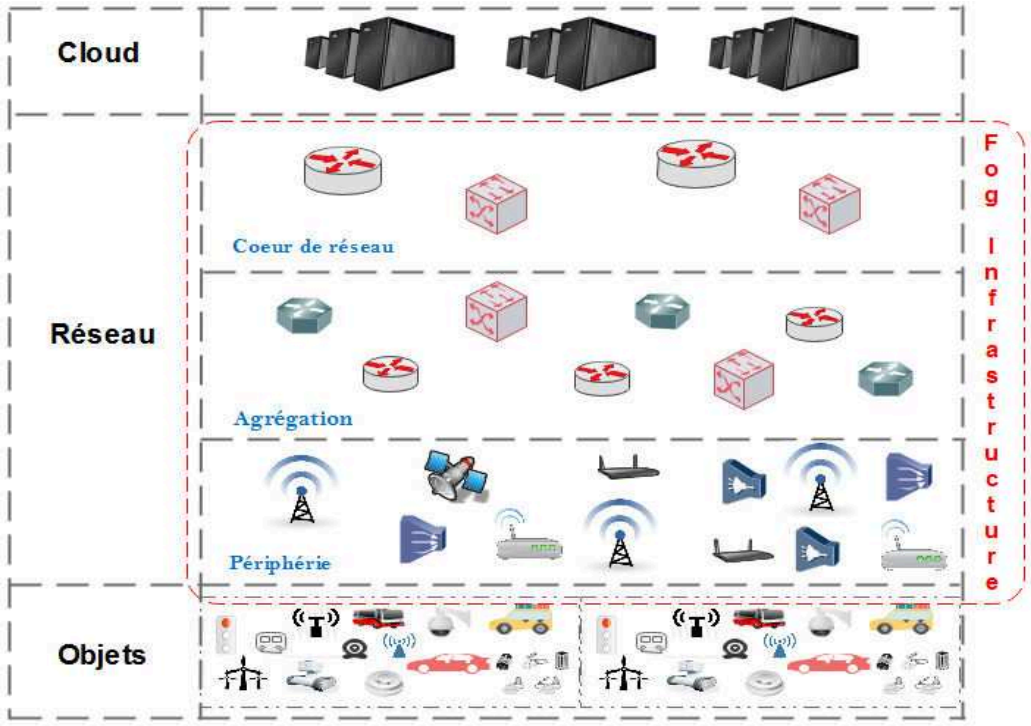


FIGURE 2 : Infrastructure du *Fog computing*.

- **Les objets :** c'est le niveau le plus proche des utilisateurs finaux et de leur environnement physique. Il inclut des objets sans fils, intelligents, mobiles

1 Les points de présence d'un opérateur sont des serveurs de collecte du réseau internet. Leur rôle est d'acheminer les données par des liens très haut débit jusqu'au cœur du réseau de l'opérateur.

ou fixes comme les capteurs, les robots, les smartphones, et les caméras connectées. Ces équipements sont géo-distribués à grande échelle. Ils sont chargés de surveiller l'environnement et transmettent des informations aux niveaux supérieurs pour l'analyse et le stockage par le biais des technologies de communication comme le Wi-Fi, Bluetooth ou le réseau cellulaire [63].

- **Le réseau :** ce niveau couvre plusieurs couches du réseau (les périphéries, l'agrégation et le cœur). Il regroupe des dispositifs de réseau comme les switchs, les routeurs et les POP. Ce niveau est utilisé pour héberger des applications nécessitant un temps de réponse court, ainsi que pour faire l'agrégation, le filtrage et le pré-traitement des données avant de les envoyer au *Cloud*. D'autres fonctionnalités peuvent être déployées dans ce niveau comme celles du CDN pour mettre en caches des données, et du SDN et du NFV pour administrer le réseau [63, 69].

- **Le *Cloud* :** c'est le niveau le plus haut dans l'infrastructure. Il s'agit de serveurs puissants distribués dans des centres de données distants permettant d'héberger des applications destinées, par exemple, à l'analyse du *Big Data* et au stockage de données permanent.

2.3.2 *Les avantages du Fog computing*

Plusieurs travaux de recherche ont listé les bénéfices apportés par le *Fog computing* par rapport au paradigme du *Cloud computing* [21, 26, 31, 42, 47, 63, 82, 85, 98]. Ci-après nous en citons quelques uns.

- *Réduction des latences :* en raison de sa proximité avec les utilisateurs finaux, le *Fog* a la capacité de supporter des applications qui nécessitent des latences courtes et stables (ex. la réalité augmentée, les jeux vidéo en ligne).

- *Réduction du trafic réseau :* le *Fog computing* permet d'exécuter des fonctions de traitement, de filtrage et d'agrégation de données tout au long du chemin du réseau. En effet, la pertinence de l'envoi de données est examinée à chaque étape de la transmission permettant une réduction importante du trafic réseau.

- *Géolocalisation et support de la mobilité :* la distribution géographique des nœuds de *Fog* aide à localiser les objets. De plus, la latence existante entre des nœuds de *Fog* voisins est courte. Cela aide à migrer des taches (ex. à base de conteneurs) d'un nœud à un autre de manière transparente, afin de supporter la mobilité des objets.

- *Passage à l'échelle :* (i) la géodistribution des nœuds de *Fog* et leur proximité avec les utilisateurs finaux permettent de gérer un grand nombre d'objets connectés (la répartition de la charge de travail). (ii) Les nœuds de *Fog* sont de nature hétérogène avec des performances et des coûts différents. Ce deuxième point permets de déployer, selon le besoin, de nouveaux nœuds de manière simple et moins coûteuse (en comparaison avec le *Cloud*).

- *Interopérabilité et fédération :* les infrastructures de *Fog* comprennent un grand nombre de nœuds géo-distribués. Plusieurs de ces nœuds peuvent fédérer/coopérer dans un cluster pour réaliser des taches complexes telle que l'analyse de données massives.

- *Décharge de traitement (offloading) :* en raison de sa proximité, le *Fog* peut aider les objets ayant des ressources limitées à décharger une partie de leurs traitements aux nœuds de calcul localisés en périphérie du réseau. Cela permet aux objets, par exemple, de préserver leur énergie, de réduire le temps de traitement ou d'augmenter la capacité de stockage de données.

- *La disponibilité :* la géo-distribution des infrastructures de *Fog* aide à lancer une tache ou à stocker plusieurs copies d'une donnée sur des nœuds de *Fog* différents. Ceci permet d'augmenter la disponibilité du service et la résilience contre la perte de données.

2.3.3 *Défis et verrous scientifiques*

Bien que le paradigme de *Fog computing* ait amené beaucoup d'avantages pour la gestion du traitement et du stockage de données, ce paradigme reste nouveau et nécessite d'investiguer les défis suivants.

1. *Hétérogénéité :* en plus de l'hétérogénéité trouvée dans l'IoT au regard des différents types d'objets, de données, de technologies de communication et de services, les infrastructures de *Fog* comprennent des nœuds de nature et de performances différentes. La gestion de l'hétérogénéité dans un environnement de *Fog* et d'IoT représente un défi majeur aujourd'hui [26].

2. *Sécurité :* les nœuds du *Fog* peuvent être déployés à l'extérieur et laissés sans surveillance (i.e. dans les rues, au-dessus des bâtiments, etc.). Ils sont donc vulnérables à des attaques telles que le détournement de données et l'écoute indiscrète [63]. Afin de préserver la sécurité du système, le déploiement des solutions pour le contrôle d'accès, l'authentification et la détection d'intrusion sont nécessaires dans chaque niveau de l'infrastructure [98, 99].

3. *Gestion et provision de ressources :* les nœuds de *Fog* sont en général des équipements réseaux ayant une puissance de calcul et une capacité de stockage limitées. Des solutions efficaces sont nécessaires pour gérer l'ordonnancement des taches dans ces infrastructures (ex. des solutions basées sur la priorité et la migration) [98]. De plus, afin de fournir un support de mobilité notamment dans le cas des objets connectés, les ressources doivent être pré-allouées, par exemple suivant des méthodes probabilistes basées sur l'historique des utilisateurs [47].

4. *Gestion des données :* le suivi et la gestion des données dans les différents niveaux de l'infrastructure du *Fog* (i.e. périphérie, agrégation et cœur) est un défi majeur. La découverte, la réplication, le placement et la persistance des données nécessitent un examen attentif dans ce contexte [86].

5. *Gestion de l'énergie :* comme mentionné ci-dessus, les infrastructures de *Fog* comprennent un grand nombre de nœuds répartis géographiquement. La consommation énergétique doit être donc plus élevée en comparaison avec celle du *Cloud* [47, 63]. De grands efforts de recherches sont nécessaires pour développer des solutions efficaces pour la gestion d'énergie, par exemple, des processus de traitement de données et des protocoles de communications moins coûteux en termes de consommation énergétique sont à développer [63].

6. *Modèle de programmation :* dans le *Cloud*, les infrastructures sont transparentes pour les utilisateurs. Les traitements sont réalisés dans des serveurs virtualisés dans les centres de données. En revanche, dans le *Fog*, les traitements sont faits à différents niveaux de l'infrastructure (i.e. périphéries, agrégation et cœur de réseau) qui sont des plates-formes dynamiques et hétérogènes. Afin de faciliter le développement des applications sur les plates-formes de *Fog computing*, il est nécessaire de fournir un modèle unifié qui prend en compte l'aspect dynamique, hiérarchique et hétérogène des ressources du *Fog* [63, 99].

7. *Qualité de service (QoS) :* dans [99], les auteurs ont étudié la QoS dans le *Fog computing* selon quatre aspects : 1) la connectivité, 2) la fiabilité, 3) la capacité et 4) la latence. Ci-après nous décrivons chacun de ces aspects en précisant les défis associés.

    a) *La connectivité :* le *Fog* étend les services du *Cloud* jusqu'aux périphéries du réseau. Dans un tel réseau hétérogène, le partitionnement, le regroupement et la collaboration fournissent une opportunité pour optimiser le coût, augmenter le débit, ou réduire la consommation énergétique [26, 99]. Le défi présenté ici consiste à concevoir des al-

gorithmes de communication efficaces pour optimiser les métriques cités ci-dessus [26].

b) *La fiabilité :* comme le *Fog computing* est réalisé par l'intégration d'un grand nombre d'équipements répartis géographiquement, la fiabilité est l'un des principaux défis à considérer lors de la conception d'un tel système [47]. La fiabilité peut être améliorée grâce à une vérification périodique pour reprendre après un échec, à un re-ordonnancement des tâches échouées ou à une réplication pour exploiter le traitement en parallèle. Toutefois, les points de contrôle et d'ordonnancement ne peuvent pas s'adapter à l'environnement hautement dynamique du *Fog* à cause des latences. La réplication semble plus prometteuse, mais elle repose sur le fonctionnement synchronisé de plusieurs nœuds de *Fog* [99].

c) *La latence :* le *Fog computing* a été proposé principalement pour lutter contre les latences élevées du *Cloud*. Le *Fog* est utilisé pour déployer des applications sensibles aux latences comme l'Internet des véhicules, la santé et l'industrie 4.0. Afin de réduire la latence, il est important d'étudier comment les données et leurs traitements sont placés dans l'infrastructure. Par exemple, un nœud de *Fog* peut avoir besoin de traiter des données distribuées dans plusieurs nœuds éloignés. Le calcul ne peut être démarré qu'après avoir récupéré toutes les données requises, ce qui ajoute de la latence au service [99].

d) *La capacité :* cet aspect a été étudié selon deux critères : (i) la bande passante réseau et (ii) la capacité de stockage. Afin d'obtenir une bande passante élevée et une utilisation efficace du stockage, il est important de réaliser des fonctions d'agrégations et de filtrage dans les périphéries du réseau. Cependant, ce niveau de l'infrastructure inclut des équipements limités en ressources de traitements et de stockage de données, ce qui crée un défi pour le choix de traitement à réaliser et de données à stocker dans ce niveau.

8. *Complexité :* les algorithmes d'optimisation existants sont en général ciblés sur le temps de traitement et l'utilisation de ressources. Vu que le *Fog* fournit une plate-forme avec des milliers de nœuds pour servir des milliards d'objets connectés, il est nécessaire de concevoir des solutions de gestion décentralisées et coopératives. Par exemple, afin d'accélérer le temps de placement de données ou de traitements, il est nécessaire d'utiliser des algorithmes parallèles ou des méthodes approchées (ex. basées sur le concept de diviser pour régner), plutôt que d'utiliser des méthodes exactes et centralisées [26].

Dans ce travail de thèse, nous nous intéressons à l'optimisation du placement données d'IoT dans les infrastructures de *Fog* minimisant la latence globale du système. Ce travail adresse les défis associés à la gestion et la provision de ressources, à la gestion de données, à l'amélioration de la QoS et à la réduction de la complexité du système.

2.4 OPTIMISATION COMBINATOIRE

L'*Optimisation Combinatoire* (OC) occupe une place très importante en recherche opérationnelle et en informatique. De nombreuses applications sont modélisées sous forme de problème d'OC comme le voyageur de commerce, le multi-sac à dos et l'ordonnancement des tâches [50].

Un problème d'optimisation combinatoire (POC) est défini par un ensemble discret et fini de solutions, et une *fonction objectif* (appelée aussi *fonction de coût*) pour évaluer chaque solution. Les meilleures solutions (ou les solutions optimales) sont celles qui minimisent (ou maximisent) la valeur de la fonction objectif. Mathématiquement, un POC peut être formulé comme suit : Soit S l'ensemble des solutions réalisables et $f : S \to \mathfrak{R}$ la fonction objectif. Un problème d'optimisation combinatoire consiste à trouver une solution $s^* \in S$ minimisant (resp. maximisant) f, c'est-à-dire, $f(s^*) = min_{s \in S}\{f(s)\}$ (resp. $f(s^*) = max_{s \in S}\{f(s)\}$).

De nombreuses méthodes ont été proposées pour résoudre les POC. Ces méthodes sont classées en deux catégories : les **méthodes exactes** qui garantissent l'optimalité de la solution trouvée, et les **méthodes approchées** qui relâchent le critère d'optimalité de la solution pour gagner en temps de résolution. Dans ce travail de thèse, nous nous intéressons particulièrement, parmi les méthodes exactes, à la programmation linéaire et, parmi les méthodes approchées, aux heuristiques et aux méta-heuristiques. Ci-après, nous donnons quelques notions sur la complexité des problèmes d'optimisation. Ensuite, nous décrivons chacune des méthodes citées ci-dessus.

2.4.1 *Complexité d'un POC*

La complexité d'un problème est la quantité de ressources (en nombre d'instructions et en espace mémoire) dont a besoin un algorithme exact pour trouver une solution optimale calculée sur une machine de Turing [20]. Un problème est de complexité polynomiale (resp. exponentiel) s'il existe un algorithme permettant de trouver une solution optimale pour toutes ses instances en un temps polynomial (resp. exponentielle) par rapport à la taille de l'instance. Un exemple de problème polynomial est celui qui consiste à trouver tous les plus courts chemins entre paires de sommets dans un graphe. L'algorithme exact trouvé par

Floyd-Warshall [53] permet de le résoudre en temps polynomial $O(n^3)$, avec n le nombre de nœuds du graphe.

Ci-après, nous citons les différents classes de complexité pour les POC.

1. **La classe** P **:** un POC ramené à un problème de décision est de classe P s'il peut être résolu, de manière exacte, par un algorithme de complexité polynomiale. On peut citer par exemple le problème de calcul de plus court chemin dans un graphe.

2. **La classe** NP **:** un POC ramené à un problème de décision est dit de classe NP si et seulement si, on peut vérifier que n'importe quelle proposition est bien une solution du problème, en un temps polynomial.

3. **La classe** $NP-complet$ **:** un POC ramené à un problème de décision est dit de classe $NP-complet$ s'il vérifie les deux propriétés suivantes : (i) il est possible de vérifier une solution en temps polynomial, et (ii) tous les problèmes de la classe NP se ramènent à celui-ci via une réduction polynomiale [50].

4. **La classe** $NP-difficile$ **:** un problème $NP-difficile$ est un problème qui remplit simplement la seconde condition (citée dans le paragraphe précédent), et donc peut être dans une classe de problème plus large et plus difficile que la classe NP.

Nous rappelons ici que le problème adressé dans cette thèse (i.e. l'optimisation du placement de données dans le *Fog*) est un problème de classe $NP-difficile$ [76].

2.4.2 *Programmation linéaire*

Un programme linéaire (PL) est un problème d'optimisation dans lequel la fonction objectif et les contraintes sont décrites par des fonctions linéaires dont les inconnues sont les variables de décision du problème. Un problème d'optimisation linéaire de n variables et m contraintes possède la forme générale suivante.

$$\begin{cases} \text{Minimiser (ou maximiser)} & \sum_{i=1}^{n} c_i.x_i \\ \text{Tel que} & \sum_{i=1}^{n} a_{1,i}.x_i \leqslant b_1 \\ & \vdots \\ & \sum_{i=1}^{n} a_{m,i}.x_i \leqslant b_m \end{cases}$$

Avec $c_i, a_{m,i}, b_m \in \Re$ et l'opérateur $\leqslant$ dans les contraintes peut-être aussi $\neq$ ou $=$.

Les variables peuvent être : (i) continues, on parle alors de la programmation linéaire, (ii) discrètes, on parle alors de la programmation linéaire en nombres

entiers (*Integer Linear Programming* ou ILP), ou (iii) mixtes, on parle alors de la programmation linéaire en nombres mixtes (*Mixed Integer Linear Programming* ou MILP).

Un problème linéaire (variables continues) peut être résolu en temps polynomial contrairement au ILP et au MILP qui sont de classe *NP-difficile*. L'algorithme de base pour résoudre les problèmes linéaires est le *Simplexe* (Dantzig 1947) [54]. Cet algorithme est combiné à d'autres méthodes, comme le *Branch&Bound* pour résoudre des problèmes MILP et ILP.

Il existe de nombreux solveurs de programmes linéaires comme : CPLEX (IBM) [6], Gurobi (Microsoft) [2], GLPK (GNU) [1] et OR-Tools (Google) [4]. Dans ce travail de thèse, nous avons utilisé CPLEX pour résoudre les problèmes linéaires que nous avons définis. Ce solveur est décrit dans la section suivante.

2.4.3 *CPLEX*

IBM ILOG CPLEX Optimization Studio [6] regroupe un ensemble d'outils pour la programmation linéaire et la programmation par contraintes. Il associe :

- un environnement de développement intégré (*Integrated Development Environment*, IDE) nommé Cplex Studio IDE (sous Windows) ou oplide (sous Linux),
- un langage de modélisation : le langage OPL (*Optimization Programming Language*),
- deux solveurs : *IBM ILOG CPLEX* pour la programmation mathématique (résolution de programmes linéaires en nombres fractionnaires, mixtes ou entiers et de programmes quadratiques) et *IBM ILOG CP Optimizer* pour la programmation par contraintes.

Actuellement, CPLEX est l'un des solveurs les plus performants disponibles. Il peut ainsi traiter des problèmes de plusieurs dizaines de millions de variables et plusieurs centaines de milliers de contraintes. Il est composé d'un exécutable et d'une bibliothèque de fonctions pouvant s'interfacer avec différents langages de programmation : C, C++, C#, Java et Python.

2.4.4 *Les heuristiques*

Une heuristique est un algorithme d'approximation qui permet de trouver souvent en temps polynomial, au moins une solution réalisable, tenant compte de la fonction objectif, mais sans assurance sur l'optimalité de la solution trouvée. En général, une heuristique est conçue pour un problème particulier, en étudiant

sa structure (heuristique adhoc). Les heuristiques peuvent être classées en deux catégories [67] :

- **Méthodes constructives :** qui construisent la solution par une suite de choix partiels et définitifs. Ces méthodes sont connues également sous le nom de *méthodes gloutonnes* (*Greedy* en Anglais)
- **Méthodes de fouille locale :** qui partent d'une solution initialement complète et, de manière répétitive, essaient d'améliorer cette solution en explorant son voisinage dans l'espace de recherche. Par exemple, les méthodes de descente substituent à la solution courante une solution voisine si et seulement si la seconde est meilleure.

2.4.5 *Les méta-heuristiques*

Les méthodes heuristiques précédentes peuvent être piégées dans un minimum local. Dans la Figure 3, une heuristique commence par une solution initial S, en explorant son voisinage, cette heuristique renvoie la solution finale qui est un *minimum local*, c'est-à-dire, une solution optimale dans cette partie de l'espace de recherche et non pas à l'échelle globale des solutions possibles. Les méta-heuristiques essaient d'échapper à ces minima locaux en acceptant temporairement des solutions qui n'améliorent pas la fonction objectif [79]. Les méta-heuristiques fournissent une méthode générale d'exploitation de l'espace des solutions et recherchent un compromis entre l'exploitation de cet espace et l'exploration par amélioration locale des solutions trouvées. Une méta-heuristique peut être adaptée pour différents types de problèmes, tandis qu'une heuristique est dédiée à un problème donné. Plusieurs méta-heuristiques sont inspirées par des systèmes naturels dans de nombreux domaines tels que : la biologie (algorithmes génétiques) la physique (recuit simulé), et aussi l'éthologie (algorithmes de colonies de fourmis) [50, 54].

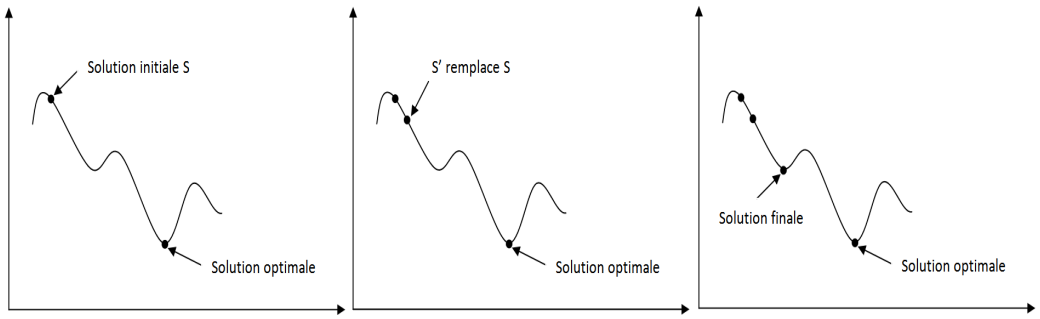


FIGURE 3 : Le problème de minimum local dans les méthodes de descente [50].

- **Les algorithmes génétiques (AG) :** sont des algorithmes inspirés de la théorie de l'évolution. Un algorithme évolutif typique est composé de trois

éléments essentiels : (1) une population constituée de plusieurs individus représentant des solutions potentielles au problème en question, (2) un mécanisme d'évaluation de l'adaptation de chaque individu de la population à l'égard de son environnement extérieur, et (3) un mécanisme d'évolution composé d'opérateurs permettant d'éliminer certains individus et de produire de nouveaux individus à partir des individus sélectionnés.

- **Le recuit simulé :** c'est une méthode basée sur la recherche locale. C'est-à-dire, l'exploration de l'espace de voisinage de la solution courante pour trouver une meilleure solution. Mais, dans ces méthodes, une solution défavorable peut-être acceptée (suivant une probabilité qui décroit au cours de l'algorithme) comme une solution courante afin d'échapper aux minima locaux.

- **Les colonies de fourmis :** ce sont des algorithmes inspirées du comportement des fourmis lorsqu'elles cherchent le plus court chemin entre leur nid et une source d'alimentation [48]. Leur principe consiste à permettre à une fourmi de choisir le plus court chemin entre le nid et la source d'alimentation en prenant le chemin le plus utilisé par les autres fourmis (à l'aide d'une phéromone). Dans cette catégorie des méta-heuristiques, chaque solution est considérée comme une fourmi se déplaçant sur l'espace de recherche. Les fourmis marquent les meilleures solutions, et tiennent compte des marquages précédents pour optimiser leur recherche.

Dans cette section, nous avons vu qu'il existait plusieurs classes de complexité pour les POC. Aussi, il existe plusieurs manière de les résoudre : exacte, heuristique et méta-heuristique. Selon la méthode de résolution utilisée, l'optimalité et le temps de recherche de la solution sont différents.

2.5 MODÈLE D'ARCHITECTURE DU SYSTÈME CONSIDÉRÉ

Dans cette section, nous décrivons le modèle d'architecture du système et donnons les notations, les définitions et les hypothèses utilisées dans les différentes contributions de ce travail de thèse.

2.5.1 *Architecture du système*

La "ville intelligente" représente un domaine majeur de l'IoT [93]. Son objectif consiste à améliorer les services et le bien être du citoyen urbain en lui fournissant des outils intelligents pour gérer le transport, la domotique, la santé, la distribution de l'énergie et de l'eau, etc. Une application de "ville intelligente"peut gérer plusieurs services parmi ceux mentionnés précédemment. Ces

services peuvent partager des données. Par exemple, les données du service de la gestion de l'énergie peuvent être utilisées par le service de domotique.

Le système considéré dans ce travail consiste en un scénario générique d'une "ville intelligente" [39] déployé dans une infrastructure de type *Fog* (voir la Figure 4). Cette infrastructure comprend un ensemble de capteurs, un ensemble de nœuds de *Fog*, un ensemble de centres de données et un ensemble d'instances de services d'IoT. Les nœuds de *Fog* sont les passerelles, les LPOP et les RPOP. Ces nœuds sont agencés hiérarchiquement : un RPOP sert un ensemble de LPOP et un LPOP sert plusieurs passerelles.

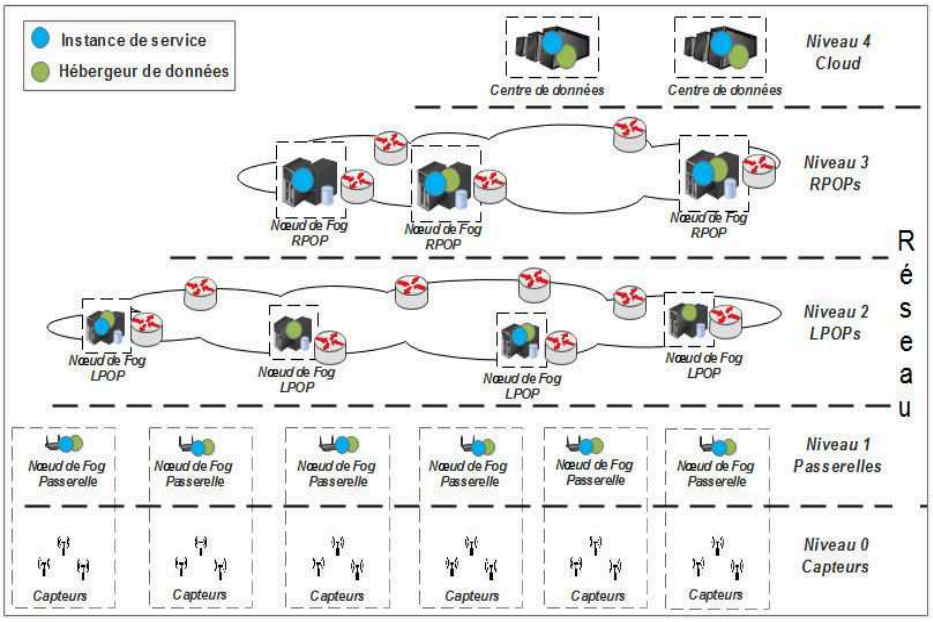


FIGURE 4 : Infrastructure du système d'un scénario d'une ville intelligente déployée dans le *Fog*.

Dans ce système, chaque instance de service est déployée dans un seul nœud de *Fog* (ou de *Cloud*) choisi par un orchestrateur d'instances de services. L'orchestrateur est une entité qui a une vision partielle ou globale de l'infrastructure (latences, performances, etc.) et gère le placement des instances de services [94]. Le choix du nœud est basé sur plusieurs critères, notamment celui des ressources disponibles, du niveau de sécurité souhaité ou de la disponibilité offerte par le nœud choisi [94].

Dans ce scénario, les capteurs considérés comme des producteurs de données collectant des informations relatives à l'environnement physique (ex. valeurs de la température). Ces informations sont envoyées aux instances de services associées via des passerelles. Ensuite, les instances de services destinataires considérées comme consommatrices de données traitent les informations envoyées pour prendre des décisions ou agir sur l'environnement (ex. éteindre le chauffage ou déclencher une alarme). Toutefois, ces instances de services peuvent produire à

leur tour d'autres données. Dans ce cas, ces instances de services sont considérées à la fois comme productrices et consommatrices de données. Les données produites dans notre système peuvent être stockées dans différents nœuds de *Fog* et centres de données de l'infrastructure. Les nœuds de stockage de données sont référencés comme des hébergeurs de données.

Dans les sections suivantes nous donnons les différentes définitions, hypothèses et notations utilisées dans ce travail de thèse.

2.5.2 *Définitions*

Définition 2.4. ***Temps de transfert*** *: c'est le temps nécessaire pour transmettre une donnée d'un nœud à un autre. Ce temps dépend de la taille de la donnée et de la bande passante du lien réseau utilisé.*

Définition 2.5. ***Latence de stockage de données*** *: c'est le temps de transfert de données depuis un nœud producteur jusqu'au nœud de stockage.*

Définition 2.6. ***Latence de récupération de données*** *: c'est le temps de transfert de données depuis un nœud de stockage jusqu'au nœud consommateur.*

Définition 2.7. ***Temps d'exécution*** *: c'est le temps pris pour le traitement de données d'une requête. Ce temps ne comprend pas le temps de transfert de données.*

Définition 2.8. ***Temps de réponse*** *[83] : appelé aussi la latence du système, c'est le temps pris par le système pour traiter une requête. Il s'agit du temps de transfert de données requis ajouté au temps d'exécution de la requête.*

Définition 2.9. ***Latence globale du système*** *: c'est la somme des temps de réponse de l'ensemble des requêtes traitées sur une période de temps donnée.*

Dans ce travail, nous avons utilisé trois classes d'acteurs du point de vue de la manipulation de données. Ces classes sont définies comme suit.

Définition 2.10. ***Hébergeur*** *(dh pour DataHost) : cette classe représente les nœuds de stockage de données qui peuvent être des nœuds de Fog ou des centres de données du Cloud. Les* dh *sont localisés dans les niveaux 1, 2, 3 et 4 de l'infrastructure du système, voir la Figure 4.*

Définition 2.11. ***Producteur*** *(dp pour DataProd) : cette classe représente les capteurs et les instances de services de l'IoT qui produisent des données. Un* dp *peut être localisé à n'importe quel niveau dans la Figure 4.*

Définition 2.12. ***Consommateur*** *(dc pour DataCons) : cette classe représente l'ensemble des instances de services qui requièrent et traitent des données. Un* dc *peut être localisé à n'importe quel niveau dans la Figure 4.*

Notons qu'un nœud de *Fog* peut être considéré à la fois comme *producteur*, *consommateur* et *hébergeur* de données.

2.5.3 *Hypothèses*

Afin de formuler le problème du placement de données de l'IoT dans le *Fog*, nous nous sommes basés sur un ensemble d'hypothèses définies comme suit.

1. Utilisation de données :
    - Les informations produites par un producteur sur une période sont considérées comme une seule donnée. Chaque producteur produit donc une seule donnée par période.
    - Les données produites par un nœud producteur peuvent être utilisées par un ou par plusieurs nœuds consommateurs (i.e. le partage de données entre plusieurs consommateurs est possible).
2. Latences
    - Les données sont toujours transmises via les chemins de latence minimale. En effet, l'envoi des données via des liens autres que ces chemins augmente potentiellement la latence globale du système, et par conséquent, s'éloigne de l'objectif de ce travail de thèse (la minimisation de la latence du système).
    - Les données sont transmises d'un nœud à un autre sous forme de paquets. Un paquet représente l'unité de base de transfert de données. La taille de cette unité de base est notée b (en octets) [83].
    - Il existe entre chaque paire de nœuds (producteur et hébergeur), une latence de base de transfert de données notée $ts_{i,j}$. Cette latence (appelée latence de stockage) représente le temps nécessaire pour transférer la quantité de données b depuis le producteur dp_i jusqu'à l'hébergeur dh_j.
    - Il existe entre chaque paire de nœuds (hébergeur et consommateur), une latence de base de transfert de données notée $tr_{k,j}$. Cette latence représente le temps nécessaire pour récupérer la quantité de données b par le consommateur dc_k depuis l'hébergeur dh_j.
3. Stockage de données :
    - Une donnée d_i produite par le producteur dp_i est stockée dans un hébergeur dh_j unique (i.e. aucune réplication ou segmentation de données n'est considérée dans le système).
    - Dans notre travail, le placement de données est fait périodiquement. La période du placement est fixée suivant un seuil de changement de l'infrastructure (ex. apparition/disparition des nœuds de *Fog*, congestion des liens réseaux ou migration des instances de service).

- Au vu de leur capacité restreinte, les capteurs ne sont pas utilisés pour stocker des données. En effet, ils transmettent leurs données aux passerelles associées.

4. Infrastructure du système :
   - Le système (l'orchestrateur de services et de données) a une vue globale de l'infrastructure et possède des informations relatives : (i) aux dépendances des flux de données entre les producteurs et les consommateurs, (ii) aux latences existantes entre les nœuds de *Fog*, (iii) aux emplacements des instances de services, et (iv) aux emplacements des hébergeurs de données ainsi que leur capacité de stockage.

2.5.4 *Notations*

Dans ce travail de thèse, nous avons utilisé plusieurs notations mathématiques pour formuler le problème du placement de données. Ces notations sont décrites dans les Tables 10 et 11 situées dans l'Annexe A.

2.6 CONCLUSION

Le présent chapitre a introduit les connaissances et les outils essentiels pour la présentation de nos contributions. Nous avons résumé les principes de base concernant l'IoT, le *Fog*, l'optimisation combinatoire et le modèle de l'infrastructure du système utilisé. Le chapitre suivant discute les travaux de l'état de l'art existants dans les domaines abordés par nos contributions.

Chapitre 3

ÉTAT DE L'ART

Nous rappelons ici que dans ce travail de thèse, nous nous intéressons à la problématique de l'optimisation du placement de données d'IoT dans une infrastructure de type *Fog*. L'objectif est de minimiser la latence globale du système.

Comme la problématique que nous cherchons à résoudre est très peu traitée dans la littérature, nous commençons ce chapitre par présenter les travaux existants pour le placement des instances de services d'IoT dans les infrastructures de *Fog*. En effet, ces travaux sont très proches de notre problématique du point de vue du contexte (i.e. *Fog computing* et IoT), des caractéristiques (i.e. l'accès partagé, la géodistribution, l'hétérogénéité, etc.) et des objectifs (i.e. placement en ligne et réduction de la latence du système). La deuxième partie de ce chapitre est dédiée à la présentation de l'état de l'art des plates-formes d'évaluation des stratégies de placement de données. Nous clôturons ce chapitre par une synthèse et conclusion.

Sommaire

Métrique / Méthode	Latence	Bande passante	Disponibilité	Énergie	Coût
Programmes linéaires	[87], [88] [46], [25] [94], [60]	[60]	[46]	[60]	[60]
Heuristiques	[90], [95] [25], [100]	[95]			
Méta-heuristiques	[72]			[72]	
Autres	[101], [95]	[101], [95]			[101]

TABLE 1 : Classification des travaux sur le placement des instances de services d'IoT dans le *Fog*. Certaines références peuvent apparaître plusieurs fois car, elles considèrent plusieurs métriques, sinon elles utilisent plusieurs méthodes de résolution. La ligne *Autres* est pour le reste des méthodes (ex. programme quadratique, algorithmes exacts, etc.).

3.1 SOLUTIONS DE PLACEMENT DES INSTANCES DE SERVICES D'IOT DANS LE *fog*

Dans la littérature, plusieurs travaux ont adressé la problématique du placement des instances de services d'IoT dans le *Fog*. Ces travaux ont été motivés par des objectifs tels que : la minimisation de la latence, la réduction de la consommation énergétique ou l'amélioration de la qualité de service. Dans la Table 1, nous avons classifié les travaux de l'état de l'art sur le placement des instances de services d'IoT dans le *Fog* selon la méthode de modélisation et de résolution en : (i) programmes linéaires, (ii) heuristiques, (iii) méta-heuristiques et (iv) autres pour le reste (ex. programme quadratique, algorithmes exacts spécifiques, etc.).

3.1.1 *Programmes linéaires*

Dans [87], les auteurs ont proposé un framework pour le provisionnement des ressources dans un environnement de *Fog computing*. Ils ont défini un modèle basé sur la programmation linéaire en nombres entiers pour optimiser le placement des tâches (ou des instances de services) dans des clusters de nœuds de *Fog* (appelés colonies de *Fog*). Chaque cluster est contrôlé par un nœud de *Fog* performant. Ce dernier joue un rôle de broker et d'orchestrateur de tâches : il reçoit des tâches à exécuter depuis différents nœuds du cluster et les place dans le cluster. Les tâches qui ne peuvent pas être placées (ex. à cause du manque des ressources) sont redirigées vers le *Cloud*. La fonction objectif de leur modèle de placement de tâches consiste à (i) maximiser le nombre de tâches exécutées à l'intérieur de chaque cluster, et par conséquent, réduire le temps de réponse de l'ensemble des tâches, et (ii) choisir pour chaque tâche le nœud de *Fog* conforme aux besoins en ressources de traitement (i.e. CPU, RAM, stockage et bande passante) d'une part, et le nœud le plus proche du contrôleur en termes de temps de latence d'autre part.

En revanche, le modèle proposé ne considère pas le temps d'exécution des tâches dans la fonction objectif. En effet, les auteurs ont supposé que les nœuds de *Fog* dans chaque cluster ont les mêmes performances. Ainsi, ils considèrent uniquement le temps de la latence de transfert existant entre le contrôleur et le nœud choisi pour placer une tâche. En réalité, les nœuds de *Fog* sont hétérogènes en termes de performances de traitement (i.e. CPU et RAM), ce qui engendre des temps d'exécution différents. C'est pour cela que les auteurs ont proposé une amélioration de leur modèle de placement des tâches dans [88] en ajoutant des contraintes sur le temps d'exécution maximal de chaque tâche.

Conduit par des contraintes de disponibilité, *Nader et al.* [46] ont formulé le problème du placement des tâches dans une infrastructure *Fog* comme un pro-

gramme linéaire en nombres entiers. Leur formulation permet de dupliquer une tâche d'un utilisateur sur plusieurs nœuds de *Fog* afin d'atteindre une disponibilité souhaitée. En effet, ils ont associé (i) une valeur de disponibilité pour chaque nœud de *Fog*, (ii) un coût de réalisation pour chaque tâche, ce coût est identique pour tous les nœuds de *Fog*, (iii) un budget maximal pour chaque utilisateur pour réaliser ses tâches, et (iv) une valeur de duplication maximale d'une tâche pour chaque utilisateur. La fonction objectif de leur modèle sert à trouver la solution de placement et de duplication de l'ensemble des tâches avec le coût minimal en respectant la disponibilité souhaitée pour chaque utilisateur.

Là encore, le coût identique peut être vu comme un point faible de leur modèle. En effet, le coût de réalisation d'une tâche est un critère générique qui peut être exprimé en latence réseau, consommation énergétique, utilisation de la bande passante, ou en termes d'autres métriques. De plus, les noeuds de *Fog* ont une nature hétérogène, notamment en ce qui concerne les performances et la latence d'accès. Cela implique des coûts différents par rapport au noeud de *Fog* choisi pour exécuter une tâche. Une amélioration possible de leur modèle consiste à associer un coût différent pour chaque paire (tâche, noeud de *Fog)*. Ce coût peut être choisi selon, par exemple, les critères cités ci-dessus.

Velasquez et al. [94] ont proposé un modèle basé sur un programme ILP pour l'optimisation du placement de services d'IoT dans le *Fog* selon trois objectifs : (i) minimiser le nombre de sauts entre l'utilisateur et l'emplacement de service demandé dans le *Fog* pour minimiser la latence, (ii) minimiser le nombre de sauts existants entre les emplacements des services coopératifs, et (iii) minimiser le nombre total de migrations de services par rapport aux emplacements antérieurs afin de maintenir le système stable. En effet, le dernier objectif limite la surutilisation du réseau en migrant un grand nombre de services (par exemple, dans le cas des infrastructures hautement dynamiques).

3.1.2 *Heuristiques*

Dans [25], les auteurs ont proposé : (i) une formulation MILP du problème de placement d'instances de services dans une infrastructure de *Fog* et (ii) plusieurs politiques basées sur des heuristiques pour le placement des instances de services. Leur idée globale consiste à placer les instances de services dans l'infrastructure de *Fog* sans l'utilisation d'un orchestrateur (global ou partiel) qui nécessite des connaissances sur l'infrastructure du système (ex. latences, performances disponibles, etc.). En effet, dans leur politique, chaque nœud du *Fog* déploie un algorithme de placement et se comporte indépendamment des autres. Afin de choisir le nœud de *Fog* qui exécute une tâche, les auteurs ont associé à chaque tâche un temps de réponse maximal. À la réception des tâches, si ce temps est satisfait par le nœud de *Fog* receveur et si ce nœud héberge une instance du ser-

vice demandé, alors la tâche est exécutée dans ce nœud. Sinon, elle est redirigée vers le nœud suivant dans la hiérarchie. Si aucun des nœuds n'est choisi, la tâche est ainsi redirigée vers le *Cloud*. Par ailleurs, pour le placement des instances de services, chaque nœud compte le nombre de tâches acceptées et rejetées, ainsi que le temps de réponse moyen requis par l'ensemble des tâches. Ensuite, suivant ces deux métriques, le nœud de *Fog* choisit l'instance de service qui sera maintenue en exécution.

En se basant sur le même principe de décentralisation de la gestion de tâches, *Gu et al.* [60] ont proposé un modèle basé sur la programmation linéaire et une heuristique pour le placement des tâches dans un environnement de *Mobile Edge Computing (MEC)*. Leur modèle optimise la consommation énergétique globale due à l'exécution et à la transmission des tâches en assurant la satisfaction des temps de réponse requis par l'ensemble des tâches. Dans leur heuristique, les nœuds serveurs (qui traitent les tâches) publient entre eux leurs performances (la fréquence du CPU). Ensuite, chaque tâche est envoyée (depuis son créateur) au nœud serveur potentiel pour l'exécution. Le choix du nœud est fait par une fonction basée sur la consommation énergétique. À la réception des tâches, il y a deux cas possibles, (i) le nœud receveur satisfait le temps de réponse requis, et par conséquent, la tâche est traitée par ce nœud, (ii) le nœud ne satisfait pas le temps de réponse requis, la tâche est alors renvoyée vers l'un des nœuds serveurs voisins en se basant sur les informations partagées des performances des nœuds. Leur algorithme de placement est distribué du fait que les tâches sont placées en s'appuyant sur des connaissances partagées entre les nœuds. Cela est fait sans l'utilisation d'un ou de plusieurs orchestrateurs de tâches qui peuvent limiter le passage à l'échelle de l'algorithme de placement.

Dans [100], les auteurs ont proposé un algorithme similaire à celui de *Gu et al.* [60] pour placer les tâches dans une infrastructure de *Fog* en minimisant la latence globale du système. L'algorithme proposé est basé sur le partage des connaissances des performances des nœuds de *Fog* pour réaliser le placement. De plus, des modèles probabilistes ont été construits pour estimer le temps d'exécution d'une tâche dans un nœud de *Fog*.

Dans [90], *Taneja et al.* ont proposé une heuristique pour placer des instances de services d'IoT dans une infrastructure de *Fog*. Leur idée consiste à trier par ordre ascendant : (i) les nœuds de *Fog* selon leur performance de traitement et (ii) les instances de services selon la quantité de ressources requises. Ensuite, leur heuristique cherche pour chaque instance de service, le nœud de *Fog* qui répond aux performances de traitement requises par cette instance. Les instances de services non placées sont redirigées vers le *Cloud*.

Bien que l'heuristique proposée soit simple et facile à déployer, elle présente plusieurs inconvénients : elle ne considère pas de contraintes sur le temps d'exécution des instances, ni sur l'équilibrage de la charge de travail entre les nœuds

de *Fog*. Par exemple, pour cette dernière, si le nœud de *Fog* le moins performant peut satisfaire les exigences de plusieurs instances de services, leur heuristique alloue toutes ces instances de services à ce nœud même s'il y a d'autres nœuds de *Fog* libres. Cependant, l'allocation des instances de services dans différents nœuds aurait permis de paralléliser leur exécution, et par conséquent, de réduire le temps de réponse de ces instances de services. Une amélioration possible consiste à choisir, pour chaque instance de service, le nœud du *Fog* le plus proche des utilisateurs en termes de latence et qui répond aux besoins en performances. En effet, cela permet d'éviter le placement des instances de services dans des nœuds qui sont très éloignés. De plus, afin de résoudre le problème de l'équilibrage de la charge de travail, plusieurs itérations sont nécessaires afin de replacer les instances de services co-localisées dans des nœuds de *Fog* libres ou dont le temps de réponse est réduit.

3.1.3 *Méta-heuristique*

Dans [72], *Mebrek et al.* ont proposé un algorithme génétique (méta-heuristique) pour l'optimisation du placement des tâches dans une infrastructure de *Fog*. Leur algorithme optimise la consommation énergétique due au transfert et à l'exécution des tâches, en respectant le temps de réponse requis pour chaque tâche. Dans leur modèle d'optimisation, le temps de réponse d'une tâche inclut le temps de transmission de la tâche, le temps de son exécution et éventuellement le temps de redirection de la requête vers le *Cloud* (dans le cas d'insuffisance de ressources). D'autre part, le taux de consommation énergétique est calculé par la somme des consommations : du transport réseau, du traitement et du stockage de la requête.

3.1.4 *Autres*

Dans [101], les auteurs ont formulé le problème du placement de services d'IoT dans le *Fog* comme un programme non linéaire respectant des contraintes sur la QoS en termes de temps de réponse. Ils ont considéré le temps de traitement, de communication, de stockage et de déploiement de service dans la fonction objectif de leur formulation. Le modèle proposé inclut des contraintes pour : (i) limiter le temps de réponse moyen de l'ensemble des services, (ii) respecter les ressources disponibles dans les nœuds de *Fog* et (iii) limiter le nombre de violations du SLA en termes de temps de réponse. Afin de résoudre le problème de placement, les auteurs ont proposé deux heuristiques. La première optimise le placement pour minimiser la violation du SLA, et la deuxième optimise le placement pour minimiser le coût opérationnel du système.

Ye et al. [95] ont proposé un modèle plus complexe pour optimiser le placement des instances de services d'IoT dans le *Fog*. La fonction objectif de leur modèle consiste à minimiser le temps de réponse moyen de l'ensemble des instances de services placées. En plus des contraintes sur la latence, les ressources disponibles et la bande passante, ils ont limité le choix des emplacements d'une instance donnée. Par exemple, afin de respecter la vie privée, les instances ne peuvent être placées que dans certains nœuds de *Fog*. Les auteurs ont proposé, en se basant sur des algorithmes de recherche avec retour sur trace (Backtrack Search Algorithm), un algorithme exact et deux heuristiques pour placer les instances de services. Cependant, les auteurs ont supposé que le temps d'exécution d'une instance ne change pas par rapport à l'emplacement choisi. Comme mentionné auparavant, les nœuds du *Fog* ont des performances hétérogènes qui influencent le temps d'exécution. De plus, une instance de service placée dans un nœud de *Fog* peut être en concurrence avec d'autres instances de service, et selon la politique d'ordonnancement mise en œuvre et le nombre de tâches existantes, le temps de réponse de cette instance peut varier.

3.1.5 *Synthèse*

Dans cette section, nous avons vu que le problème du placement des instances de services d'IoT dans le *Fog* a été étudié en utilisant plusieurs méthodes. Des chercheurs ont proposé des programmes linéaires pour résoudre ce problème et d'autres ont proposé des heuristiques afin de réduire le temps de placement dans les grandes infrastructures. Ces heuristiques sont basées sur deux approches : (i) celles qui utilisent un ou plusieurs orchestrateurs de services et (ii) celles qui n'en utilisent pas. L'avantage de la première approche est que ces méthodes peuvent réaliser un placement efficace, mais elles reposent sur des connaissances de l'état du système qui sont difficiles à collecter. À l'inverse, les méthodes de la deuxième approche dans laquelle chaque nœud se comporte indépendamment des autres réalisent un placement rapide et sans point unique de défaillance, mais le manque de connaissances sur l'état de système peut conduire à un mauvais placement (ex. problème d'équilibrage de charge). Aussi, d'autres auteurs ont proposé des algorithmes exacts pour placer les services. Mais, au vu de la grande complexité du système, ces algorithmes sont inapplicable dans la réalité. Par ailleurs, les méthodes de placement basées sur des méta-heuristiques sont très peu proposées dans ce contexte. Cela revient, selon nous, à la nouveauté du paradigme de *Fog computing* qui est toujours en phase d'initiation.

Dans cette thèse, nous proposons trois stratégies pour le placement de données de l'IoT dans les infrastructures de *Fog*, sous la forme d'une approche exacte et deux heuristiques. Dans ces stratégies, nous mettons en œuvre des orchestrateurs qui placent les données dans une ou dans plusieurs partitions de l'infra-

structure d'une manière indépendante afin de réduire le temps de résolution. Le problème d'équilibrage de charge entre les noeuds de *Fog* (voir le paragraphe discutant le travail [90]) est traité au niveau des partitions par les orchestrateurs de données car ces derniers ont une vision sur les partitions gérées. Aussi, la dépendance du coût de placement vis-à-vis des nœuds choisis (voir le paragraphe discutant le travail [46]) est également pris en charge dans nos solutions. En revanche, nous n'avons pas considéré de borne sur les transmissions des données, cela peut néanmoins être rajouté à notre modèle de programme linéaire.

3.2 Outils d'évaluation des solutions de placement de données dans le *fog*

Dans cette section, nous résumons les travaux de l'état de l'art sur la gestion du placement de données dans le contexte du *Fog* et de l'IoT. La Table 2 illustre une classification des travaux existants sur la mise en œuvre des environnements de *Fog* et d'IoT selon les quatre critères suivants :

- la gestion du placement des données
- la prise en compte de l'environnement de *Fog*
- la modélisation des objets connectés
- le type de l'outil proposé parmi S (simulation), E (émulation) et R (plate-forme réelle).

Nous rappelons ici que chaque méthode a des avantages et des inconvénients. Par exemple, les simulations sont moins coûteuses en termes de temps d'évaluation et de budget de mise en œuvre mais leurs résultats ne sont pas précis. L'émulation est plus coûteuse mais plus précise que la simulation, alors que les plates-formes réelles sont très coûteuses mais sont plus précises en termes de résultats.

3.2.1 *Plate-forme réelle*

Dans [23], *Adjih et al.* ont proposé *FIT-IoT-LAB*, une plate-forme expérimentale fédérée à grande échelle pour la conception et la comparaison des protocoles, applications et services d'IoT. Elle comprend 2700 capteurs physiques sans fil dont 117 nœuds de robots mobiles déployés sur six sites en France. Cette plate-forme n'intègre ni nœuds de *Fog* ni stratégies de placement de données.

3.2.2 *Émulateur*

Concernant les outils basés sur l'émulation, *Mayer et al.* ont proposé *EmuFog* [71], et *Coutinho et al.* ont introduit *Fogbed* [43], deux émulateurs d'environnements de *Fog*. Les deux considèrent des infrastructures de *Fog* comprenant des switchs et des routeurs mais ils n'émulent pas d'objets connectés. Cependant, *Fogbed* utilise la simulation pour manipuler les objets connectés. De plus, ces deux émulateurs n'intègrent pas la gestion du placement de données. En raison de la grande densité des infrastructures de *Fog*, les solutions basées sur l'émulation peuvent être très coûteuses en termes de ressources et de temps d'émulation (plusieurs semaines, voir plusieurs mois).

3.2.3 *Simulateur*

GridSim [89] est un simulateur d'environnements pair-à-pair et de *Grilles de calcul*. Il permet de modéliser et de simuler des ressources de grille et des réseaux, et d'évaluer des solutions d'allocation et d'ordonnancement des tâches, et enfin de gérer le placement de données dans l'infrastructure simulée. Cependant, il n'y a pas de support dédié aux environnements de *Fog* et d'IoT dans *GridSim*. Ainsi, les utilisateurs doivent définir des modèles de nœuds de *Fog* et des objets d'IoT afin de mettre en œuvre leurs stratégies de placement de données dans ce contexte.

Etemad et al. [52] ont proposé un simulateur d'environnements de *Fog* afin de pouvoir valoriser l'impact de l'utilisation du *Fog* en terme de latence de service. Leur simulateur simule des centres de données et des nœuds de Fog, mais il ne simule ni les objets connectés ni la gestion du placement des données dans l'infrastructure simulée.

TABLE 2 : Classification des travaux sur les outils d'évaluation des stratégies du placement de données dans le *Fog* et l'IoT (S : simulation, E : émulation, R : plate-forme réelle).

Travail	**Placement de données**	***Fog***	**IoT**	**Plate-forme**
GridSim [89]	+	-	-	S
[52]	-	+	-	S
IoT-LAB [23]	-	-	+	R
EmuFog [71]	-	+	-	E
Fogbed [43]	-	+	+	E
Fog-Torch [32]	-	+	+	S
iFogSim [62]	-	+	+	S

Dans [32], les auteurs ont introduit *FogTorch*. Il s'agit d'un prototype de simulateur d'environnements de *Fog* et d'IoT. Il a été utilisé principalement dans [32] pour la mise en œuvre d'un algorithme pour trouver le déploiement d'une application d'IoT dans une infrastructure de *Fog*. Ce déploiement est contraint par diverses exigences de qualité de service en termes de latence et de bande passante. *FogTorch* prend en entrée les spécifications (un modèle statique) de l'infrastructure de *Fog*, l'application à déployer et la QoS demandée en termes de latence et de bande passante, et produit en sortie la solution du déploiement qui répond à ces exigences. En revanche, la simulation du comportement des environnements de *Fog* et d'IoT n'est pas encore déployée dans *FogTorch* (lors de l'écriture de ce manuscrit) : la simulation de l'envoi, du traitement et du stockage de données n'a pas été mise en œuvre dans ce simulateur. Ces fonctionnalités sont nécessaires pour évaluer le comportement dynamique des politiques des utilisateurs (ex. pour le placement de données) suivant les différents changements d'état.

Dans [62], *Gupta et al.* ont proposé *iFogSim*, un simulateur d'environnements de *Fog* et d'IoT. Ce logiciel simule des infrastructures avec des millions d'objets connectés et des milliers de nœuds de *Fog* (et de centres de données de *Cloud*). Il permet aux utilisateurs de définir et de mettre en œuvre leurs solutions de gestion des ressources pour le placement et l'ordonnancement des services d'IoT dans l'infrastructure simulée, et de mesurer l'impact en termes de latence de service, de congestion du réseau, de consommation énergétique et de coût opérationnel. Cependant, la gestion du placement de données n'est pas mise en œuvre dans ce simulateur.

3.2.4 *Synthèse*

Au vu des caractéristiques d'*iFogSim* (passage à l'échelle, modélisation des environnement de *Fog* et d'IoT), nous avons choisi d'adapter ce simulateur pour mettre en œuvre et évaluer nos stratégies de gestion du placement de données. De plus, ce simulateur a été utilisé dans le cadre de nombreux travaux dans le contexte du *Fog* et de l'IoT [30, 75, 87, 88]. Dans le chapitre 6, nous proposons une extension de ce simulateur permettant de déployer et d'évaluer des stratégies de gestion du placement de données dans le contexte de *Fog* et d'IoT.

3.3 CONCLUSION

Dans ce chapitre d'état de l'art, nous avons présenté dans un premier temps des travaux d'optimisation du placement des instances de services d'IoT dans les infrastructures de *Fog*. Ces travaux se concentrent principalement sur la réduction de la latence du système, mais cela peut être accompagné par d'autres objec-

tifs comme la réduction de la consommation énergétique et le coût opérationnel ou encore l'amélioration de la QoS. Ces travaux se basent généralement sur la programmation linéaire pour formuler le problème du placement et ils utilisent éventuellement des heuristiques et des méta-heuristiques pour trouver une solution approchée dans un temps acceptable. Ensuite, nous avons présenté des travaux de l'état de l'art sur les outils potentiels pour l'évaluation des stratégies de placement de données dans le contexte du *Fog computing* et de l'IoT.

Après avoir présenté le contexte et les travaux de la littérature liés à notre étude, nous présentons dans la partie suivante l'ensemble de nos contributions.

Deuxième partie

CONTRIBUTIONS

Chapitre 4

IFOGSTOR : UNE STRATÉGIE DE PLACEMENT DE DONNÉES DE L'IOT DANS LES INFRASTRUCTURES FOG

Le *Fog computing* est un paradigme qui consiste à étendre les services fournis par le *Cloud* jusqu'aux bordures du réseau. Ce paradigme utilise les ressources des équipements localisés dans le réseau (ex. les routeurs et les switchs) pour stocker et traiter des données. Le *Fog* présente une infrastructure dense, géo-distribuée et hétérogène ; ses équipements ont des performances de calcul, capacités de stockage et latences réseaux différentes. Ces éléments ont une grande influence sur la latence du système. Cela pose un problème : le mauvais choix des emplacements de stockage de données augmente considérablement la latence d'accès aux données.

Dans ce chapitre, nous présentons *iFogStor* ; notre première contribution pour le placement de données de l'IoT dans les infrastructures de type *Fog*. *iFogStor* offre deux possibilités pour placer les données ; (i) une méthode exacte qui trouve une solution optimale minimisant la latence globale du système, et (ii) une méthode à base d'heuristique qui trouve une solution approchée à l'optimale pour un placement en ligne dans le cas de grandes infrastructures. Nous commençons ce chapitre par présenter la formulation du problème de placement de données dans les infrastructures de type *Fog*. Ensuite, nous décrivons les deux méthodes proposées pour placer les données. La méthodologie et quelques résultats d'évaluation d'*iFogStor* sont présentés dans la dernière partie de ce chapitre.

Sommaire

4.1 PROBLÉMATIQUE

Dans les infrastructures de type *Fog*, les services de l'IoT (i.e. les unités de traitement de données) sont placés selon plusieurs objectifs [72, 88, 90], par exemple, pour optimiser le temps de réponse du système, respecter un niveau de sécurité, équilibrer la charge sinon minimiser la consommation énergique. Dans le contexte de l'IoT, plusieurs services éloignés les uns des autres peuvent partager des données provenant de la même source. Cela nécessite de stocker ces données dans des nœuds de telle façon à minimiser les latences d'accès. D'autre part, les nœuds de *Fog* ont des capacités de stockage limitées, ce qui ne leur permet pas de stocker toutes les données qui leurs sont affectées. De plus, un service d'IoT peut demander des données de plusieurs sources différentes. En effet, la récupération des données requises depuis leur source augmente considérablement le temps de latence du service. La problématique adressée dans ce chapitre consiste à trouver comment placer les données de l'IoT dans des infrastructures de type *Fog* tout en minimisant la latence globale du système, c'est-à-dire la somme des latences de l'ensemble des transferts de données générées.

Les infrastructures de *Fog* sont très dynamiques et leur topologie de réseau change fréquemment. Cela implique de replacer les données périodiquement et ce, en cours d'exécution. Cependant, vu le grand nombre de nœuds de *Fog* et la quantité massive de données à gérer, le problème de placement de données est de classe *NP-difficile* [76]. Ceci nécessite un temps de résolution très important. La deuxième problématique adressée dans ce chapitre consiste à trouver comment accélérer le temps de résolution du problème de placement afin de replacer les données périodiquement et en cours d'exécution.

4.2 APERÇU D'IFOGSTOR

Cette section présente un aperçu d'*iFogStor*, la stratégie que nous proposons pour placer les données de l'IoT dans les infrastructures de *Fog* permettant de minimiser la latence globale du système.

La Figure 5 schématise l'infrastructure de notre système, voir la Section 2.5 du chapitre 2. Cette infrastructure comprend un ensemble de capteurs (ex. capteurs de température), un ensemble de nœuds de *Fog*, un ensemble de centres de données et un ensemble d'instances de services d'IoT.

Dans ce système, les capteurs (producteurs de données) considérés collectent des informations relatives à l'environnement physique (ex. valeurs de la température). Ces informations sont envoyées aux instances de services associées via des passerelles. À la réception, les instances de services (consommateurs de données) traitent les informations envoyées pour prendre des décisions ou agir

sur l'environnement (ex. éteindre le chauffage ou déclencher une alarme). Toutefois, ces instances de services peuvent produire à leur tour d'autres données (i.e. producteurs et consommateurs de données). Les données produites dans notre système peuvent être stockées dans différents nœuds de *Fog* et centres de données (hébergeurs de données) de l'infrastructure.

Dans cette infrastructure, une instance de service est déployée dans un seul nœud de *Fog* (ou de *Cloud*) choisi par un orchestrateur d'instances de services selon plusieurs critères (ex. ressources disponibles, niveau de sécurité, etc.).

Dans un tel système, une approche naïve pour réduire la latence globale du système consiste à placer les données à coté de chaque consommateur. Cependant, cette approche présente plusieurs problèmes. Premièrement, les données peuvent être partagées entre plusieurs consommateurs éloignés géographiquement. Une solution possible consiste à faire plusieurs réplications de la donnée, et exige donc de gérer la cohérence de données. De plus, l'envoi de plusieurs réplications à différents endroits augmente le trafic et peut saturer le réseau. Deuxièmement, les consommateurs de données peuvent migrer (ou être créés et supprimés dynamiquement) nécessitant de replacer les données selon les nouveaux emplacements des consommateurs. Finalement, les nœuds de *Fog* ont des capacités de stockage limitées. Le choix des données à stocker dans chaque nœud est crucial pour réduire la latence de récupération de ces données.

Dans cette contribution, nous proposons une approche qui prend en considération les problèmes cités ci-dessus (i.e. partage de données, limitation des capa-

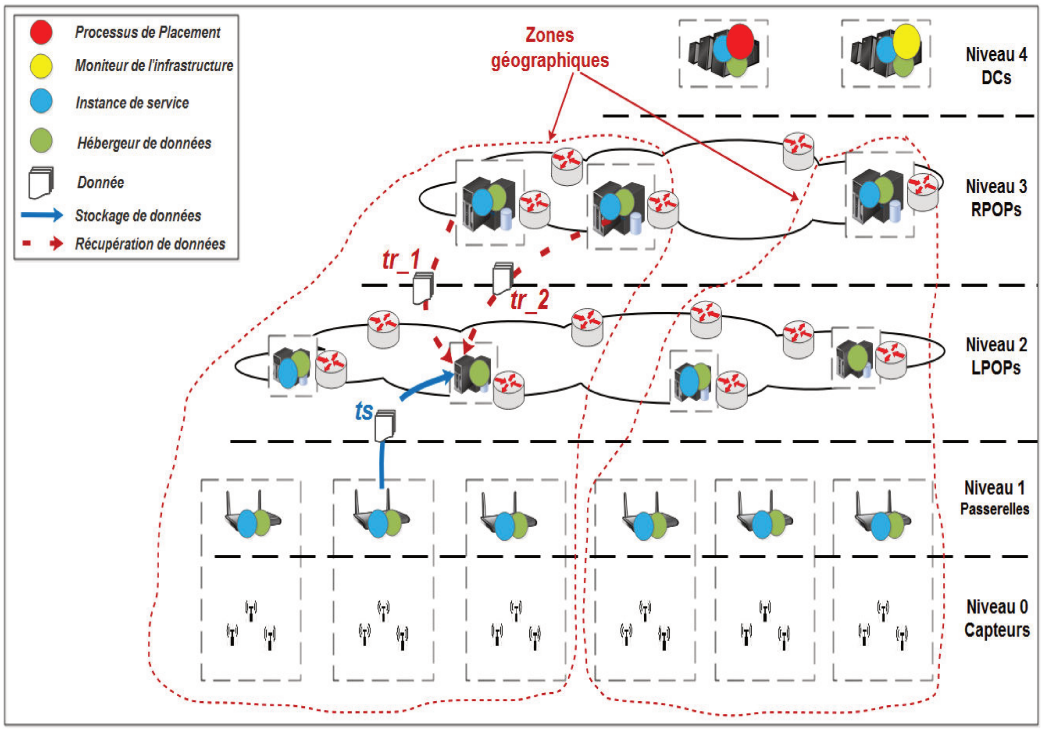


FIGURE 5 : Architecture du système.

cités de stockage) et qui trouve la solution la plus appropriée pour stocker les données. Notre stratégie détermine, pour toutes les données, les emplacements de stockage (i.e. hébergeurs de données) qui minimisent la latence globale de transfert des données dans le système, voir la Section 2.5 du chapitre 2. En effet, les données sont stockées et ensuite récupérées avec une latence de transfert qui dépend de l'emplacement de stockage choisi. Cette latence est composée de deux temps de transfert ; (i) temps de transfert de la donnée depuis le producteur vers l'emplacement de stockage (notée *ts* dans la Figure 5), et (ii) le temps de transfert de la donnée depuis l'emplacement de stockage vers les consommateurs (notée *tr* dans la Figure 5).

Comme le problème que nous adressons consiste à placer un ensemble de données dans un ensemble d'emplacements sachant que : (i) les données sont de tailles différentes, (ii) les emplacements ont des capacités différentes, et (iii) chaque pair (donnée, emplacement) a un coût de latence différent, nous avons modélisé ce problème comme un problème d'affectation généralisé (Generalized Assignment Problem, GAP en Anglais) [29]. Le problème GAP est détaillé dans la section suivante.

4.3 PROBLÈME D'AFFECTATION GÉNÉRALISÉ

GAP [29] est un problème répandu dans la littérature sur l'optimisation combinatoire [28, 29, 36, 41]. Ce problème consiste à trouver une affectation d'un ensemble de n objets sur un ensemble de m emplacements (par exemple, des tâches sur des processeurs) qui minimise ou maximise le coût global de cette affectation. Les emplacements ont des capacités différentes notées $\{c_1, c_2, ..., c_m\}$, et chaque objet a une taille différente et un coût différent selon l'emplacement choisi. Les tailles des objets et les coûts de placement sont notés respectivement $\{s_{1,1}, ..., s_{1,m}, ..., s_{n,1}, ..., s_{n,m}\}$ et $\{v_{1,1}, ..., v_{1,m}, ..., v_{n,1}, ..., v_{n,m}\}$. $s_{i,j}$ représente la taille de l'objet i par rapport à l'emplacement j, et $v_{i,j}$ le coût de l'affectation de l'objet i à l'emplacement j. La solution choisie doit respecter deux contraintes : (i) chaque objet est affecté à un seul et unique emplacement, et (ii) les capacités des emplacements ne sont pas dépassées [29]. La formulation linéaire du problème GAP est décrite comme suit :

$$\begin{cases} \text{Minimiser} & \sum_{i=1}^{n}\sum_{j=1}^{m} v_{i,j}.x_{i,j} \\ \text{Sous contraintes.} & \\ & \sum_{j=1}^{m} x_{i,j} = 1 \qquad \forall i \in [1..n] \\ & \sum_{i=1}^{n} s_{i,j}.x_{i,j} \leqslant c_j \quad \forall j \in [1..m] \\ & x_{i,j} \in \{0,1\} \qquad \forall i \in [1..n], \forall j \in [1..m] \end{cases}$$

$x_{i,j} = 1$ signifie que l'objet *i* est affecté à l'emplacement *j*, sinon $x_{i,j} = 0$.

Dans les sections suivantes, nous décrivons notre formulation et notre approche pour résoudre le problème de placement de données de l'IoT dans les infrastructures de *Fog*. Nous rappelons ici que les notations, les définitions et les hypothèses utilisées dans cette formulation sont décrites dans la Section 2.5 du chapitre 2.

4.4 MODÈLE DE PLACEMENT DE DONNÉES

Dans cette section, nous donnons notre formulation du problème de placement de données de l'IoT dans le *Fog*.

4.4.1 *Formulation du problème*

Considérons le système S composé d'un ensemble d'hébergeurs noté $DH = \{dh_1, dh_2, ..., dh_n\}$, un ensemble de producteurs noté $DP = \{dp_1, dp_2, ..., dp_l\}$ et un ensemble de consommateurs noté $DC = \{dc_1, dc_2, ..., dc_m\}$. DP produit une quantité de données notée $D = \{d_1, d_2, ..., d_l\}$, tel que la donnée d_i est produite par le producteur dp_i. Chaque donnée d_i est de taille s_{d_i} (en octets). L'ensemble D de données est stocké dans DH et consommé par DC.

Soit A_D la matrice d'affectation de D dans DH. Le coefficient $x_{i,j} = 1, x_{i,j} \in A_D$ signifie que la donnée d_i est affectée à l'emplacement de stockage dh_j, sinon $x_{i,j} = 0$. Le but de ce travail consiste à trouver A_D en minimisant la latence globale du système.

$$A_D = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{l,1} & \cdots & x_{l,n} \end{bmatrix}, x_{i,j} \in \{0,1\} \tag{1}$$

Soit TSD la matrice des latences de transfert de la quantité de données D depuis les producteurs DP vers les emplacements de stockage DH. Le coefficient

$ts_{d_{i,j}}$ indique le temps de transfert de la donnée $d_i \in D$ depuis son producteur dp_i jusqu'à l'emplacement de stockage $dh_j \in DH$.

$$TSD = \begin{bmatrix} ts_{d_{1,1}} & \cdots & ts_{d_{1,n}} \\ \vdots & \ddots & \vdots \\ ts_{d_{l,1}} & \cdots & ts_{d_{l,n}} \end{bmatrix} \tag{2}$$

Dans le système S, chaque donnée produite est utilisée par un ou par plusieurs consommateurs. Soit DFM la matrice des flux de données qui relie chaque producteur de données à l'ensemble des consommateurs associés. Le coefficient $f_{i,k} = 1$ signifie que les données du producteur dp_i sont utilisées par le consommateur dc_k. En effet, chaque consommateur $dc_k \in DC$ récupère une quantité de données depuis chaque emplacement de stockage $dh_j \in DH$. Soient $D_{k,j} \subset D$ la quantité de données à récupérer par le consommateur dc_k depuis l'emplacement de stockage dh_j, et $tr_{D_{k,j}} \in TRD$ son temps de transfert (i.e. le temps nécessaire pour récupérer $D_{k,j}$ par dc_k depuis dh_j). TRD est la matrice des temps de récupération de toutes les données dans le système S.

$$DFM = \begin{bmatrix} f_{1,1} & \cdots & f_{1,m} \\ \vdots & \ddots & \vdots \\ f_{l,1} & \cdots & f_{l,m} \end{bmatrix}, f_{i,k} \in \{0,1\} \tag{3}$$

$$TRD = \begin{bmatrix} tr_{D_{1,1}} & \cdots & tr_{D_{1,n}} \\ \vdots & \ddots & \vdots \\ tr_{D_{m,1}} & \cdots & tr_{D_{m,n}} \end{bmatrix} \tag{4}$$

Le stockage de D dans DH et ensuite sa récupération par DC génèrent la latence globale de transfert de données suivante :

$$Latence\_Globale = \sum_{ts_{d_{i,j}} \in TS_D} ts_{d_{i,j}}.x_{i,j} + \sum_{tr_{D_{k,j}} \in TR_D} tr_{D_{k,j}} \tag{5}$$

Nous supposons que la latence globale de transfert est égale à la somme des latences de transfert de toutes les données.

Soit FC le vecteur des capacités de stockage libre existant dans les emplacements de stockage, tel que f_{dh_j} est la capacité de stockage libre existante dans l'emplacement dh_j :

$$FC = \{f_{dh_1}, f_{dh_2}, ..., f_{dh_n}\} \tag{6}$$

4.4.2 *Contraintes*

Dans cette formulation, nous avons défini deux contraintes :

1. La quantité de données affectée à chaque hébergeur de données $dh_j \in DH$ doit être inférieure ou égale à sa capacité de stockage libre f_{dh_j} :

$$\forall f_{dh_j} \in FC : \sum_{i \in [1..l]} s_{d_i}.x_{i,j} \leqslant f_{dh_j} \tag{7}$$

2. Chaque donnée d_i est affectée à un hébergeur de données dh_j unique. Par conséquent, la somme des valeurs de chaque ligne dans A_D doit être égale à 1 :

$$\forall i \in [1..l] : \sum_{j \in [1..n]} x_{i,j} = 1 \tag{8}$$

4.4.3 *Modèle de la latence du système*

Dans cette partie, nous définissons le modèle de la latence globale de transfert de données dans notre système. Ce modèle est utilisé pour évaluer le placement de données spécifié dans la matrice A_D.

Comme mentionné auparavant, la latence globale de transfert de données est égale à la somme des latences de stockage de données dans DH et des latences de récupération de données par l'ensemble des consommateurs DC, voir l'équation (5).

Premièrement, nous définissons un modèle pour calculer la latence de stockage de données $ts_{d_{i,j}} \in TS_D$ générée par le transfert de la donnée d_i depuis son producteur dp_i jusqu'à l'emplacement de stockage dh_j, voir ts dans la Figure 5. Cette latence est calculée par le produit du nombre d'unités de base, b, à transférer et le temps de transfert d'une unité de base :

$$ts_{d_{i,j}} = \left\lceil \frac{1}{b}.s_{d_i} \right\rceil .ts_{i,j} \tag{9}$$

Deuxièmement, nous définissons un modèle pour calculer la latence de récupération des données $tr_{D_{k,j}} \in TR_D$ générée par le consommateur dc_k en récupérant les données requises depuis l'emplacement de stockage dh_j jusqu'à son nœud de déploiement, voir tr_1 et tr_2 dans le Figure 5. Cette latence est calculée par le produit du nombre d'unités de base, b, à récupérer depuis l'emplacement de stockage dh_j et le temps de transfert d'une unité de base :

$$tr_{D_{k,j}} = \left\lceil \frac{1}{b}.s_{D_{k,j}} \right\rceil .tr_{k,j} \tag{10}$$

$s_{D_{k,j}}$ décrit la taille des données requises, qui est calculée par :

$$s_{D_{k,j}} = \sum_{i \in [1..l]} s_{d_i}.f_{i,k}.x_{i,j} \tag{11}$$

l'équation (11) consiste à additionner les tailles des données s_{d_i} qui sont stockées dans dh_j ($x_{i,j} = 1$) et consommées par le dc_k ($f_{i,k} = 1$).

En substituant l'équation (11) dans l'équation (10), nous obtenons le modèle final pour la latence de récupération des données :

$$tr_{D_{k,j}} = tr_{k,j}.(\left\lceil \frac{1}{b}.s_{d_1} \right\rceil .f_{1,k}.x_{1,j} + ... + \left\lceil \frac{1}{b}.s_{d_l} \right\rceil .f_{l,k}.x_{l,j}) \tag{12}$$

En remplaçant l'équation (12) et l'équation (9) dans l'équation (5), nous obtenons le modèle final de la latence globale de transfert de données dans le système :

$$Latence\_Globale = \sum_{i \in [1..l]} \sum_{j \in [1..n]} \alpha_{i,j}.x_{i,j} \tag{13}$$

Avec $\alpha_{i,j} = \left\lceil \frac{1}{b}.s_{d_i} \right\rceil . \left(ts_{i,j} + \sum_{k \in [1..m]} tr_{k,j}.f_{i,k} \right)$, $\alpha_{i,j}$ définit la valeur de la latence de transfert générée par le placement de la donnée d_i dans l'emplacement de stockage dh_j.

L'objectif de la stratégie de placement que nous proposons consiste à trouver l'affectation de D dans DH qui minimise la latence globale de transfert de données dans le système S. Cela revient à trouver la matrice définie dans l'équation (1) qui minimise l'équation (13). Cette problématique est modélisée par le système linéaire suivant :

$$\begin{cases} \text{Minimiser} & \sum_{i \in [1..l]} \sum_{j \in [1..n]} \alpha_{i,j}.x_{i,j} \\ \text{Sous contraintes.} & \\ & \sum_{i \in [1..l]} s_{d_i}.x_{i,j} \leqslant f_{dh_j} \quad \forall j \in [1..n] \\ & \sum_{j \in [1..n]} x_{i,j} = 1 \quad \forall i \in [1..l] \\ & x_{i,j} \in \{0,1\} \quad \forall i \in [1..l], \forall j \in [1..n] \end{cases}$$

4.5 SOLUTIONS POUR LE PLACEMENT DE DONNÉES PROPOSÉES

Après avoir modélisé le problème du placement de données de l'IoT comme un problème de type GAP en utilisant des méthodes basées sur la programmation

linéaire, nous proposons dans cette section deux solutions pour résoudre ce problème. La première, dénommée *iFogStor*, est une solution exacte qui trouve les emplacements optimaux pour stocker les données en minimisant la latence globale du système. La seconde solution, dénommée *iFogStorZ*, est une heuristique basée sur le concept "de diviser pour régner"afin de réduire le temps de calcul du placement de données dans les grandes infrastructures. *iFogStorZ* trouve une solution approchée à l'optimale pour placer les données en cours d'exécution. Dans les deux solutions, nous utilisons le solveur de systèmes linéaires *CPLEX* MILP [6]. Les deux solutions sont détaillées ci-après.

4.5.1 *Solution exacte : iFogStor*

Cette solution consiste à résoudre le problème de placement de données comme un seul programme linéaire en utilisant le solveur *CPLEX* MILP. Elle détermine les emplacements idéaux pour stocker toutes les données en minimisant la latence globale du système. Cependant, comme mentionné auparavant, le problème de placement de données est un problème *NP-difficile* [28]. Par conséquent, cette solution peut engendrer un problème d'explosion combinatoire en temps de résolution. Or, le placement de données doit être fait en cours d'exécution afin de replacer les données périodiquement suivant les changements de l'infrastructure. Par exemple, parmi ces changent nous citons l'apparition/disparition de nœuds de *Fog* et la migration des services (voir la Section 4.1 de ce chapitre).

4.5.2 *Solution heuristique : iFogStorZ*

Afin d'accélérer le temps de calcul du placement de données dans les grandes infrastructures (i.e. infrastructures comprenant des milliers de nœuds de Fog), *iFogStorZ* aborde le problème de placement en subdivisant l'infrastructure en plusieurs zones géographiques. Pour chaque zone, un sous-problème de placement de données avec une complexité réduite est modélisé et résolu avec une méthode exacte. Les sous-problèmes peuvent être traités indépendamment et éventuellement en parallèle, afin de réduire le temps global de placement. Cette méthode se base sur l'hypothèse que : Dans le contexte de l'IoT, les données sont en général utilisées dans les régions dans lesquelles elles sont produites. Par exemple, dans les applications de "ville intelligente", domotique, ou Internet des véhicules, les données sont traitées localement afin de déclencher des événements ou générer des notifications rapides. De plus, la logique du *Fog computing* consiste à rapprocher les traitements des données autant que possible des producteurs. Les services sont donc censés être déployer près des sources de données, c'est-à-dire, dans les mêmes zones géographiques.

Dans *iFogStorZ*, une partition géographique (ou zone) contient tous les nœuds de *Fog* localisés dans la même région géographique (ex. ville, région, etc.). Pour chaque zone, un sous-problème de placement de données est modélisé et résolu séparément des autres zones suivant la méthode exacte *iFogStor*. Les solutions des sous-problèmes sont agrégées pour construire la solution globale de placement. Cette méthode permet de réduire le temps de placement de données. En effet, le partitionnement du problème de placement qui est *NP-difficile* en sous-problèmes réduit significativement sa complexité, et par conséquent, le temps de placement. De plus, la possibilité de résoudre les sous-problèmes séparément et en parallèle réduit également le temps de placement. Cette solution peut permettre de recalculer le placement de données uniquement pour une zone de l'infrastructure.

Cependant, le partitionnement géographique entraîne une perte d'information dans l'espace de recherche de la solution. Plus précisément, il s'agit d'une perte sur la localisation des instances de service consommatrices localisées dans des zones autres que celles des sources de données. Dans ce cas, les données ne sont pas placées d'une manière optimale. L'accroissement du nombre de zones génère plus de perte d'information, et par conséquent, une dégradation de la qualité du placement plus importante. Le compromis existant entre la réduction du temps de placement par l'augmentation du nombre de zones et la perte en qualité de la solution est étudié dans la Section 4.6.4.

Plusieurs définitions sont possibles pour une zone dans *iFogStorZ*. Dans ce travail de thèse, nous avons choisi les POP régionaux (RPOP) [14] comme points de partitionnement de l'infrastructure. Ces équipements sont localisés dans le niveau 3 de l'infrastructure du système, voir la Figure 5. Ainsi, le nombre maximal de partitions possibles est égal au nombre total de RPOP. Plusieurs RPOP peuvent être regroupés pour former une zone. Par exemple, dans la Figure 5, le problème est partitionné en deux zones dont la première contient deux RPOP et la deuxième ne contient qu'un seul RPOP.

4.6 ÉVALUATION ET RÉSULTATS

Dans cette section, nous présentons dans un premier temps le scénario et les différentes configurations utilisées dans nos expérimentations. Ensuite, nous décrivons la méthodologie utilisée pour évaluer les deux solutions proposées pour placer les données dans les infrastructure de *Fog*. Nous finissons cette section par discuter des résultats d'expérimentations.

4.6.1 *Scénario d'expérimentation*

Afin d'évaluer notre stratégie de placement de données, nous considérons le scenario générique de "ville intelligente"décrit dans la Section 2.5 du chapitre 2. Nous rappelons que dans ce scenario, il existe un ensemble de capteurs (ex. humidité, température et luminosité) qui surveillent l'état physique de l'environnement et transmettent leurs données à un ensemble de services de l'IoT déployés dans le *Fog* et dans le *Cloud* (voir la Figure 5). Les nœuds de *Fog* sont constitués de passerelles, de LPOP et de RPOP. Ces nœuds sont structurés hiérarchiquement : un RPOP sert un ensemble de LPOP et un LPOP sert plusieurs passerelles. Dans nos expérimentations, nous avons fixé le nombre de capteurs gérés par une passerelle à 100. Notons que, les capteurs transfèrent leurs données seulement aux passerelles associées. Les données des capteurs ne nécessitent donc pas une optimisation de placement.

La Figure 6 montre un exemple de dépendances de données (flux de données) existantes entre les instances de services de ce scénario. La génération et la consommation de données se passent comme suit. (i) Les capteurs collectent des données relatives à l'environnement et les transmettent aux instances de services localisées dans les passerelles. (ii) Ces dernières traitent les données relevées et renvoient les données résultantes à une ou à plusieurs instances de services (i.e. instances de consommateurs). Dans nos expérimentations, nous avons fait varier le nombre d'instances de consommateurs qui partagent les mêmes données de 1 à 5. Ces instances sont choisies aléatoirement parmi l'ensemble d'instances localisées dans la même zone géographique. (iii) De la même manière, les instances localisées dans les LPOP traitent les données reçues depuis les passerelles, et renvoient les données résultantes aux instances de services localisées dans les RPOP et dans les centres de données dans le *Cloud*. (iv) Les instances localisées dans les RPOP traitent les données reçues (i.e. données de passerelles et LPOP) et renvoient leurs résultats aux centres de données dans le *Cloud*. (v) Les instances de services localisées dans les centres de données traitent les données reçues et sauvegardent les résultats localement pour archivage.

4.6.2 *Outils et configurations*

Dans nos expérimentations, nous avons utilisé *iFogSim* [62] pour simuler le scénario proposé. *iFogSim* est un simulateur d'environnements de *Fog* et d'IoT. Il permet de mesurer et d'évaluer la latence du système, le trafic réseau, la consommation énergétique et le coût opérationnel d'une application. Nous avons étendu *iFogSim* afin de pouvoir mettre en œuvre et évaluer notre stratégie de placement. L'extension que nous avons proposée effectue trois actions : (i) formuler linéairement le problème de placement de données, (ii) utiliser le solveur *CPLEX* MILP

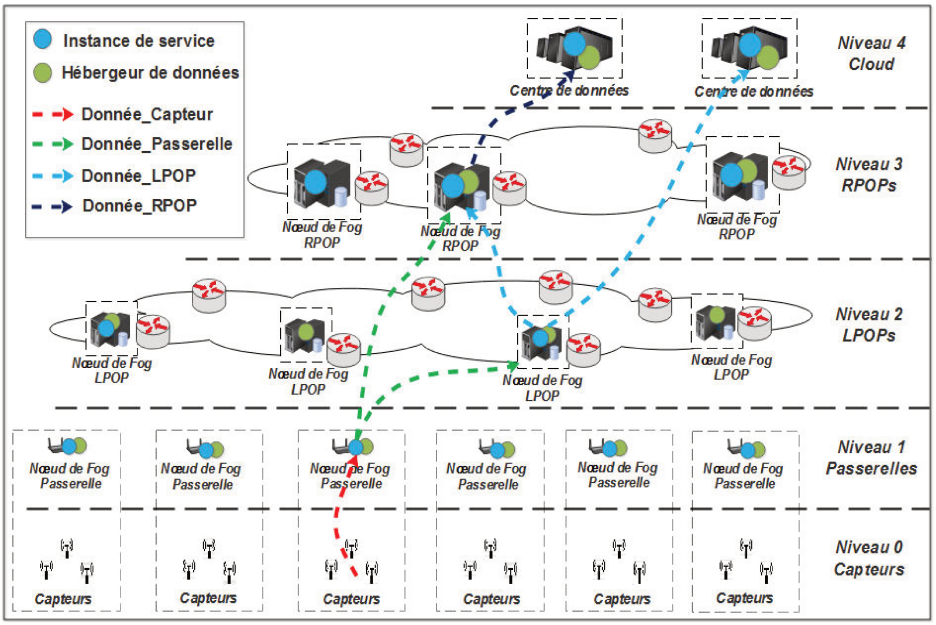


FIGURE 6 : Les flux de données dans le scénario expérimental.

pour résoudre le problème linéaire et (iii) récupérer la solution puis configurer le placement de données. Cette extension est décrite plus en détails dans le chapitre 6.

La Figure 7 montre le diagramme de séquence des événements entre *iFogSim* et *CPLEX*. Premièrement, *iFogSim* génère l'infrastructure du système (i.e. création d'instances de services, nœuds de *Fog*, centre de données, capteurs, dépendances de données, etc.). Ensuite, il génère le problème de placement dans un fichier texte envoyé en entrée de *CPLEX* MILP. Une fois le problème résolu, *iFogSim* récupère le résultat, assigne le placement de données et commence la simulation pour mesurer la latence du système.

Comme mentionné auparavant, chaque *hébergeur* a une capacité de stockage limitée, et entre chaque paire de nœuds dans le réseau il existe une latence. La Table 3 montre la capacité de stockage libre initiale dans chaque type d'*hébergeur*, et la Table 4 illustre la valeur de la latence pour les liens physiques du réseau [74].

Nous supposons que les échanges de données se font à base de paquets. La taille d'un paquet varie entre un minimum de 34 octets et un maximum de 65550 octets [83]. Dans nos expérimentations, nous avons fixé la taille d'une donnée produite par un capteur à 96 octets [37], et celle qui est produite par une instance de service à 960 octets.

Nous avons fixé l'infrastructure à 5 centres de données, 10 RPOP, 50 LPOP, et nous avons fait varier le nombre de passerelles de 1000 à 7000 avec un pas de 1000 passerelles (afin de varier la complexité du système) ce qui représente plus qu'une ville. En effet, dans [40], les auteurs ont utilisé 1200 passerelles pour représenter une ville.

TABLE 3 : Capacités de stockage totales.

Hébergeur	Capacité de stockage simulée
Passerelle	100 Go
LPOP	10 To
RPOP	100 To
Centre de données	10 Po

TABLE 4 : Latences.

Lien réseau	Latence simulée (ms)
IoT - Passerelle	10
Passerelle - LPOP	50
LPOP - RPOP	5
RPOP - Centre de données	100
RPOP - RPOP	5
Centre de données - Centre de données	100

Les expérimentations ont été réalisées en utilisant un *RuggedPoD* [15], un calculateur contenant un CPU de 32 cœurs *Xeon E5-2620* cadencés à 3,0 GHz et de 96 Go de *RAM*, utilisant la distribution *Ubuntu* version 14.04 du système d'exploitation *GNU/Linux*.

4.6.3 *Méthodologie d'évaluation*

Afin d'évaluer notre stratégie de placement de données, nous avons utilisé deux métriques : (i) la latence globale générée dans le système (voir l'équation (13)), et (ii) le temps nécessaire pour calculer le placement de données (i.e. le temps de résolution du problème).

Nous avons étudié ces deux métriques en considérant deux modes de stockage de données : *Cloud* et *Fog*. Dans le premier mode, toutes les données produites dans le système sont stockées dans des centres de données dans le *Cloud*. Ce mode illustre la méthode traditionnelle pour stocker les données de l'IoT. Dans le deuxième mode, les données sont stockées dans le *Fog* suivant trois stratégies de placement de données :

1. *Stratégie naïve* : dans cette stratégie, le système stocke les données produites dans les nœuds de stockage les plus proches de leur producteur en termes

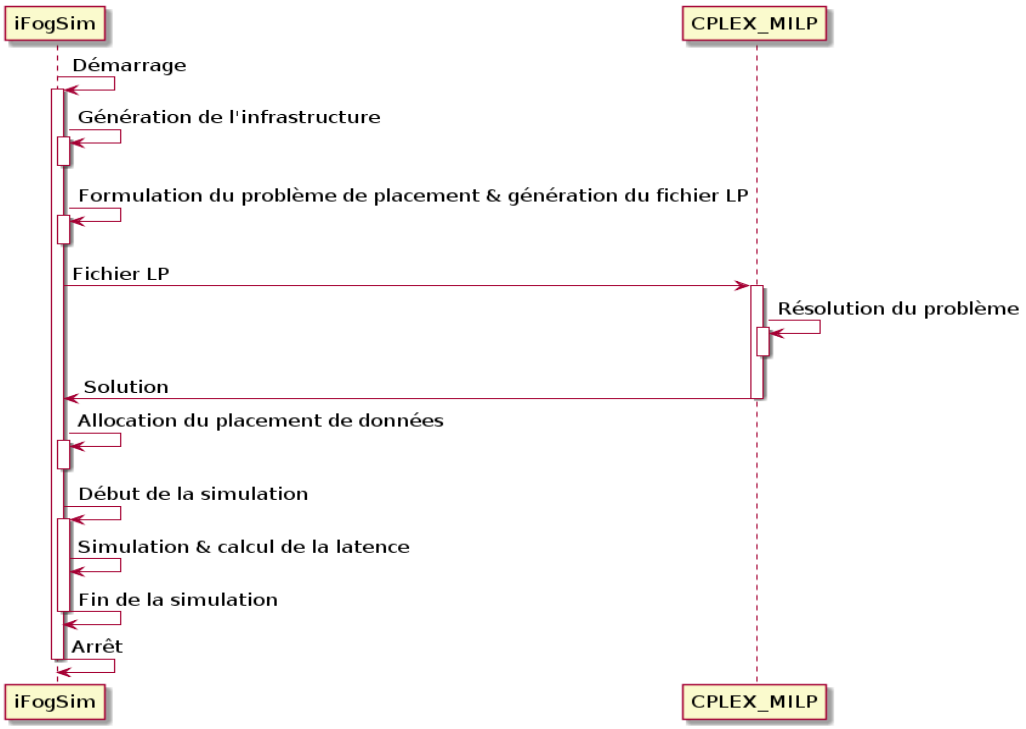


FIGURE 7 : Diagramme de séquence entre *iFogSim* et *CPLEX*.

de latence réseau. Cette méthode ne prend pas en compte la latence liée à la consommation de données.

2. *iFogStor* : les données générées sont stockées suivant notre approche exacte qui trouve les emplacements optimaux pour stocker les données en minimisant la latence globale du système.

3. *iFogStorZ* : les données générées sont stockées suivant notre heuristique qui trouve les emplacements qui minimisent la latence dans une zone géographique. Comme mentionné auparavant, cette méthode présente un compromis entre le gain en réduction du temps de placement et la perte en qualité de placement due à la génération des flux de données inter-zone lors du partitionnement de l'infrastructure. Pour cela, nous avons choisi comme nombre de zones : 2, 5 et 10 (notés *iFogStorZ_2*, *iFogStorZ_5* et *iFogStorZ_10*), et pour chaque cas, nous avons évalué le temps du calcul de placement et la perte en qualité par rapport à une solution optimale.

4.6.4 *Résultats et discussion*

Dans cette section, nous analysons des résultats de nos expérimentations.

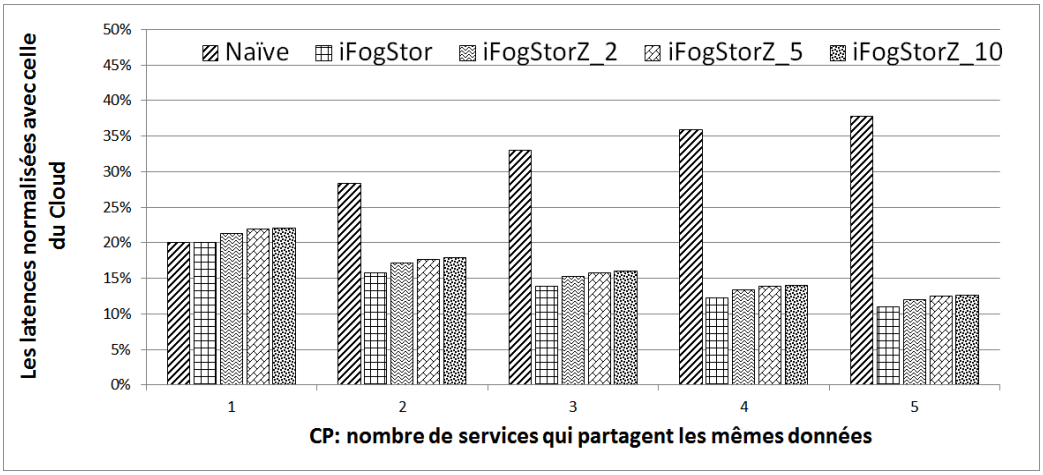


FIGURE 8 : Valeurs de latence avec 1000 passerelles.

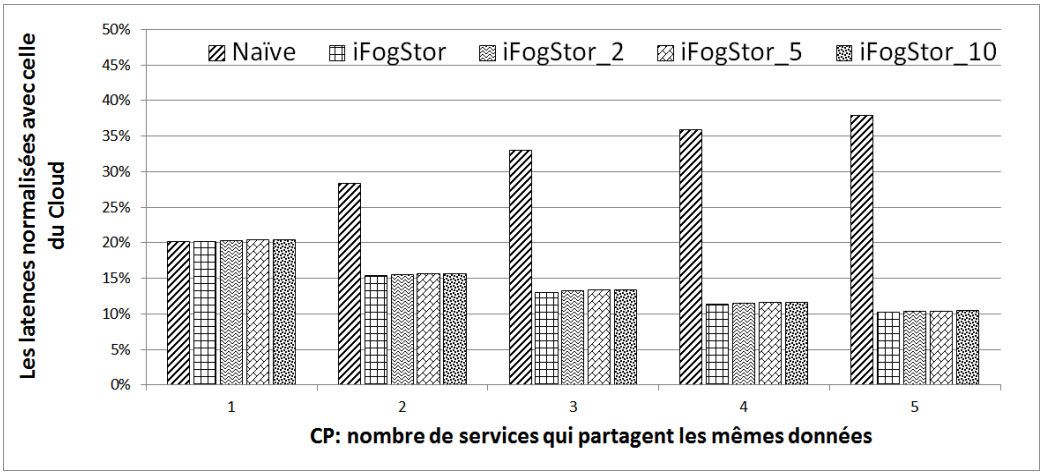


FIGURE 9 : Valeurs de latence avec 7000 passerelles.

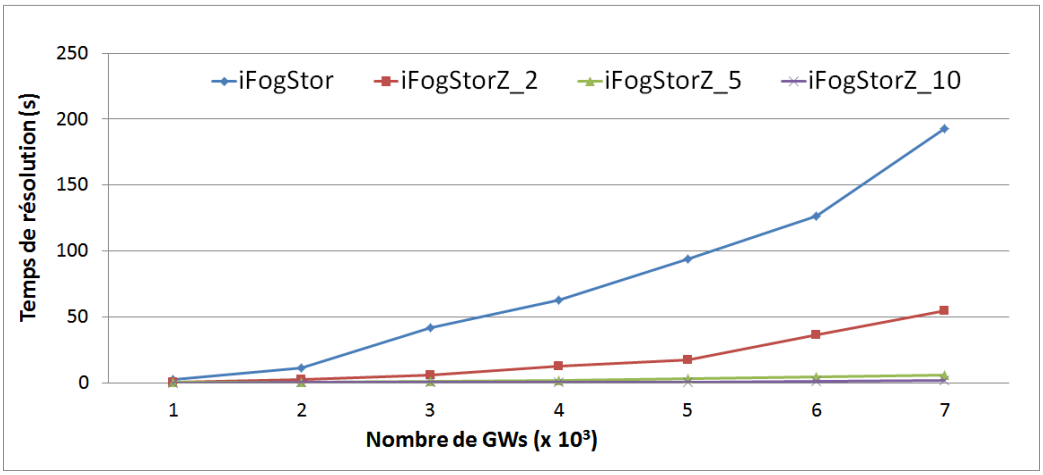


FIGURE 10 : Temps du calcul de placement.

TABLE 5 : Comparaison des stratégies de placement au regard de la qualité.

Stratégie de stockage	**Qualité de placement en (%)**
Naïve vs Cloud	67,06
iFogStor vs Cloud	87,02
iFogStor vs Naïve	60,60
iFogStorZ_2 vs iFogStor	-2,39
iFogStorZ_5 vs iFogStor	-3,41
iFogStorZ_10 vs iFogStor	-3,77

TABLE 6 : Réduction en temps de placement par rapport à *iFogStor*.

Heuristique	**Accélération**
iFogStorZ_2	4,62
iFogStorZ_5	28,29
iFogStorZ_10	117,91

4.6.4.1 *Latence globale du système*

Les Figures 8 et 9 montrent la valeur de la latence globale générée en utilisant les différents stratégies de stockage, respectivement pour une infrastructure de 1000 passerelles et une infrastructure de 7000 passerelles. Ces valeurs sont normalisées par rapport à celles obtenues par l'utilisation de la stratégie de stockage *Cloud* qui sert de référence. Notons que les différentes configurations d'infrastructures évaluées (i.e. de 1000 à 7000 passerelles) présentent approximativement les mêmes performances, c'est pour cela que seules les configurations de 1000 et 7000 passerelles sont montrées.

Nous observons que la latence générée en utilisant le stockage dans le *Cloud* est très élevée en comparaison avec les autres stratégies de stockage. La réduction en latence est proche de 80% quand *cp =1* pour tous les autres stratégies de stockage. Cela est dû à la distance existante entre les instances de services et les emplacements de stockage de données dans les centres de données. De plus, cette stratégie génère un grand trafic réseau par l'envoi et la récupération des données vers et depuis les centres de données (i.e. depuis les dp vers les dh et ensuite depuis les dh vers les dc).

En revanche, par l'utilisation de la stratégie de placement *Naïve*, la latence du système est réduite en comparaison avec celle du *Cloud* (ex. par 80% quand il n'y a pas de partage de données). Cela illustre l'avantage de l'utilisation du paradigme du *Fog computing*.

Cependant, dans le cas du partage de données entre plusieurs instances de services (i.e. quand $cp > 1$), la stratégie de placement *Naïve* perd en performances et génère des latences élevées. En effet, la latence globale du système croît respectivement avec l'augmentation du nombre d'instances qui partagent les mêmes données (noté par cp). Par exemple, les performances sont diminuée de 80% à 62% quand $cp = 5$. En effet, cette stratégie de stockage ne prend pas en considération le partage de données entre les instances de services.

En utilisant *iFogStor*, la latence a été significativement diminuée en comparaison avec celles du *Cloud* et de la stratégie de placement *Naïve*. Comme montré dans la Table 5, la latence globale du système a été diminuée en moyenne par plus de 87% en comparaison avec le *Cloud* et par plus de 60% en comparaison avec la stratégie de placement *Naïve*. Notons que, quand $cp = 1$ (i.e. pas de partage de données), la stratégie de placement *Naïve* et *iFogStor* présentent les mêmes performances. Dans ce cas, chaque donnée est stockée proche de son unique consommateur, pour les deux stratégies de stockage. Cependant, quand la valeur de cp augmente, contrairement à la stratégie de placement *Naïve*, *iFogStor* présente de meilleures performances en termes de qualité de placement. Par exemple, quand $cp = 5$, la latence a été réduite par plus de 89% en comparaison avec le *Cloud* et par plus de 73% en comparaison avec la stratégie de placement *Naïve* (voir la Table 5).

Comme montré dans la Table 5, l'heuristique *iFogStorZ* présente de bonnes performances (pour les jeux de données testés) quel que soit le nombre de zones géographiques utilisées. Le gain en latence (qualité de placement) est légèrement diminué par l'augmentation du nombre de zones. En effet, la perte en qualité de placement est de 2,39% par l'utilisation de 2 zones, et de 3,77% par l'utilisation de 10 zones. Ces valeurs peuvent être augmentées en variant le type de workload testé. En effet, elles sont propositionnelles avec le taux d'utilisation des données entre des zones différentes (i.e. la génération d'une perte d'information plus importante par le partitionnement de l'infrastructure). Nous allons détailler cela dans le chapitre 5.

4.6.4.2 *Temps du calcul de placement*

La Figure 10 montre le temps pris par le solveur *CPLEX* MILP pour résoudre le problème de placement de données pour les deux méthodes proposées : (i) *iFogStor*, l'approche exacte, et (ii) *iFogStorZ*, l'heuristique basée sur le partitionnement de l'infrastructure avec différents nombres de zones.

Nous observons qu'en utilisant *iFogStor*, le temps nécessaire pour résoudre le problème de placement augmente exponentiellement avec l'augmentation du nombre d'hébergeurs. En effet, Le solveur *CPLEX* MILP a pris environ une se-

conde pour résoudre un problème de 1000 passerelles, et 200 secondes pour résoudre un problème de 7000 passerelles (i.e. 7065 hébergeurs[1]).

Nous observons aussi qu'avec l'utilisation de l'heuristique *iFogStorZ*, le temps de résolution a été significativement réduit avec l'augmentation du nombre de zones. En effet, l'accélération en temps de placement est d'un facteur 4,62 en utilisant 2 zones et 117,91 en utilisant 10 zones (voir la Table 8) par rapport à la solution exacte. Cela s'explique par la réduction de la complexité en subdivisant le problème original en sous-problèmes. En effet, le temps de résolution a été réduit de 200 secondes à 1,4 seconde en utilisant 10 zones. Ces résultats confortent la possibilité de déployer *iFogStorZ* en ligne.

4.7 CONCLUSION

Dans ce chapitre, nous avons présenté notre formulation du problème de placement de données de l'IoT dans les infrastructures de type *Fog*. Nous avons proposé deux solutions pour résoudre le problème de placement. La première est une solution exacte qui trouve les emplacements de stockage optimaux qui minimisent la latence globale du système. Cependant, cette méthode peut causer un problème d'explosion combinatoire en temps de résolution pour les grandes infrastructures. La deuxième est à base d'heuristique réduisant le temps de résolution du problème de placement. Cette heuristique utilise une approche de type "diviser pour régner", en subdivisant l'infrastructure du problème en plusieurs zones géographiques constituant plusieurs sous-problèmes de complexités moindres qui sont résolus indépendamment les uns des autres. Cette solution présente deux avantages : (i) l'accélération du temps de résolution du problème en réduisant la complexité des sous-problèmes, et par la possibilité de résoudre les sous-problèmes en parallèle, et (ii) l'isolation qui aide à ne calculer le placement que pour une zone donnée (ex. une zone active). En revanche, cette solution peut engendrer des pertes significatives en qualité de placement pour les applications hautement distribuées (i.e. les données sont utilisées par des consommateurs situées dans des zones différentes que les producteurs). De plus, cette solution ne passe pas l'échelle du fait de la limitation en nombre de zones géographiques distinctes.

Le chapitre 5 présente une deuxième heuristique permettant de résoudre les problèmes cités plus haut. Cette heuristique est basée sur des méthodes de modélisation et de partitionnement de graphes.

1 7000 passerelles + 50 LPOP + 10 RPOP + 5 centres de données.

Chapitre 5

IFOGSTORG : UNE HEURISTIQUE BASÉE SUR LE PARTITIONNEMENT DE GRAPHES POUR LE PLACEMENT DES DONNÉES DANS LE FOG

L'heuristique *iFogStorZ* présentée dans le chapitre 4 subdivise l'infrastructure du système géographiquement sans tenir compte des flux de données existants entre les différentes partitions. Le fait de ne pas considérer ces flux induit une perte d'information sur l'espace de recherche de la solution et, par conséquent, dégrade l'optimalité[1] de la solution (voir la Section 1.3 du Chapitre 1). De plus, les partitions géographiques peuvent être déséquilibrées en termes de complexité (i.e. en nombre de nœuds et en quantité de données). En effet, la différence en complexité augmente le temps global de placement, car ce dernier dépend de la taille de la partition la plus grande. Dans ce chapitre, nous proposons *iFogStorG*, une heuristique basée sur la modélisation et le partitionnent de graphe pour subdiviser l'infrastructure du système. L'objectif d'*iFogStorG* est de minimiser la perte d'information lors du processus de partitionnement et de fournir plus de flexibilité et d'améliorer le passage à l'échelle. Nous commençons ce chapitre par donner un aperçu d'*iFogStorG*. Ensuite, nous décrivons les différentes étapes de la conception de l'heuristique *iFogStorG*. Enfin, une discussion des résultats est présentée.

Sommaire

1 L'optimalité d'une solution décrit sa qualité vis à vis à la solution exacte.

5.1 PROBLÉMATIQUE

Comme mentionné auparavant, le *Fog computing* présente une infrastructure très dynamique nécessitant un placement de données périodique et en cours d'exécution. Une méthode possible pour réduire le temps de placement de données dans le *Fog* est d'utiliser une approche basée sur le concept de diviser pour régner. Cette méthode décompose le problème original qui a une complexité exponentielle (de classe *NP-difficile*) en plusieurs sous-problèmes qui sont résolus séparément et éventuellement en parallèle. En revanche, cette méthode peut causer une perte d'information sur les emplacements des services consommateurs, et par conséquent, dégrade la qualité de placement. De plus, les parties générées peuvent être déséquilibrées en termes de complexité. Ceci augmente le temps global de placement dans l'infrastructure. Dans ce chapitre, nous allons répondre à la problématique suivante : comment pouvons-nous partitionner l'infrastructure de *Fog* en plusieurs sous-parties équivalentes en minimisant la perte d'information sur les emplacements de services afin d'achever un placement réduisant la latence du système et faisable en temps exécution ?

5.2 APERÇU D'IFOGSTORG

Considérons un système composé d'un ensemble de nœuds de *Fog* et un ensemble d'instances de services. Une instance de service peut être productrice et/ou consommatrice de données. Nous nous référons aux nœuds de *Fog* dédiés au stockage de données comme hébergeurs de données.

Comme mentionné précédemment, nous supposons que les instances de services sont déjà placées dans l'infrastructure, et le système dispose des informations relatives : (i) aux emplacements des instances de services, (ii) aux flux de données existants entre les producteurs et les consommateurs de données, (iii) aux latences réseaux existantes entre les différents nœuds de *Fog*, et (iv) aux emplacements des hébergeurs de données ainsi que leur capacité de stockage (voir la Section 2.5 du chapitre 2).

Notre objectif consiste à réduire la complexité du problème de placement de données dans ce système et de rendre sa résolution faisable en ligne. En effet, les infrastructures de *Fog* sont très dynamiques, ce qui nécessite une relocalisation périodique des données. Pour ce faire, nous utilisons une approche basée sur le concept "diviser pour régner" [49] en se basant sur la théorie des graphes. Notre idée consiste à partitionner l'infrastructure du système en plusieurs partitions constituant plusieurs sous-problèmes de placement avec des complexités équivalentes en maintenant, autant que possible, les producteurs et leurs consommateurs associés dans les mêmes partitions. Ensuite, une stratégie

de placement de données est lancée pour chaque partition. Enfin, les solutions des sous-problèmes sont agrégées pour construire la solution globale.

Le processus de partitionnement dans notre approche assure 2 propriétés :

- **Les partitions doivent être équivalentes :** le temps de résolution d'un problème de placement de données dans le *Fog* augmente exponentiellement avec la taille du problème. Par conséquent, afin de réduire le temps de résolution, les tailles des sous-problèmes de placement dans les différentes parties de l'infrastructure doivent être approximativement équivalentes. D'autre part, la complexité d'un problème de placement de données dépend du nombre d'hébergeurs de données et de la quantité de données à placer. Or, ces éléments doivent être répartis équitablement entre les différentes partitions de l'infrastructure afin de produire des partitions de complexités équivalentes.

- **Les partitions doivent être disjointes :** cette contrainte assure que les partitions doivent être indépendantes en termes d'échange de données. C'est-à-dire, les données générées dans une partition ne sont consommées que par des consommateurs localisés dans celle-ci, dans la mesure du possible. En effet, l'existence de flux de données inter-partition induit une perte de la qualité de placement de données car les consommateurs se trouvant sur d'autres partitions ne sont pas considérés. Par conséquent, les données ne seront pas placées d'une manière optimale et le système génère ainsi une latence élevée. Afin d'éviter de telles pertes, les flux de données existants entre les différentes partitions de l'infrastructure doivent être minimisés. Autrement dit, les producteurs et les consommateurs associés doivent être autant que possible maintenus dans les mêmes partitions lors du processus de subdivision de l'infrastructure.

Pour cette heuristique, nous avons modélisé l'infrastructure du système sous forme de graphe non-orienté. Ce graphe est pondéré en nœuds et en arêtes. Les poids des nœuds sont utilisés pour équilibrer la complexité (et par conséquent le temps de traitement) du problème de placement entre les différentes partitions de l'infrastructure. Les poids des arêtes sont utilisés pour minimiser les flux de données inter-partition. Cette heuristique est basée sur 4 étapes décrites dans la Figure 11.

1. **Modélisation de l'infrastructure :** nous avons modélisé l'infrastructure du système sous forme de graphe non-orienté. Les nœuds du graphe sont utilisés pour modéliser les différents nœuds de *Fog* existants dans l'infrastructure du système, et les arêtes sont utilisées pour modéliser les liens physiques entre les différents nœuds de *Fog*, voir la Figure 11 (a) et (b). Ce graphe est pondéré en nœuds et en arêtes dans l'étape 2.

2. **Pondération du graphe :** les poids des nœuds du graphe sont calculés en comptant le nombre de producteurs existants dans les nœuds de *Fog*. Les poids des arêtes sont calculés en comptant le nombre de flux de données passant par les liens physiques du réseau associé.

3. **Partitionnement du graphe :** une fois le graphe pondéré, ce dernier est partitionné en K sous-graphes en appliquant la méthode de *k-way partitioning* [73]. L'objectif de cette méthode est de minimiser la somme des poids des arêtes déconnectées (coupées) en respectant la contrainte selon laquelle les poids des nœuds du graphe original doivent être approximativement répartis entre les K sous-graphes. Cela permet de minimiser le nombre de flux de données inter-partition générés, et équilibre ainsi le temps de placement entre les K sous-parties de l'infrastructure.

4. **Placement de données :** dans cette étape, l'infrastructure du système est subdivisée selon le résultat du partitionnement du graphe. Chaque partie de l'infrastructure est traitée de façon indépendante pour placer les données en utilisant *iFogStor*, et éventuellement en parallèle avec d'autres parties.

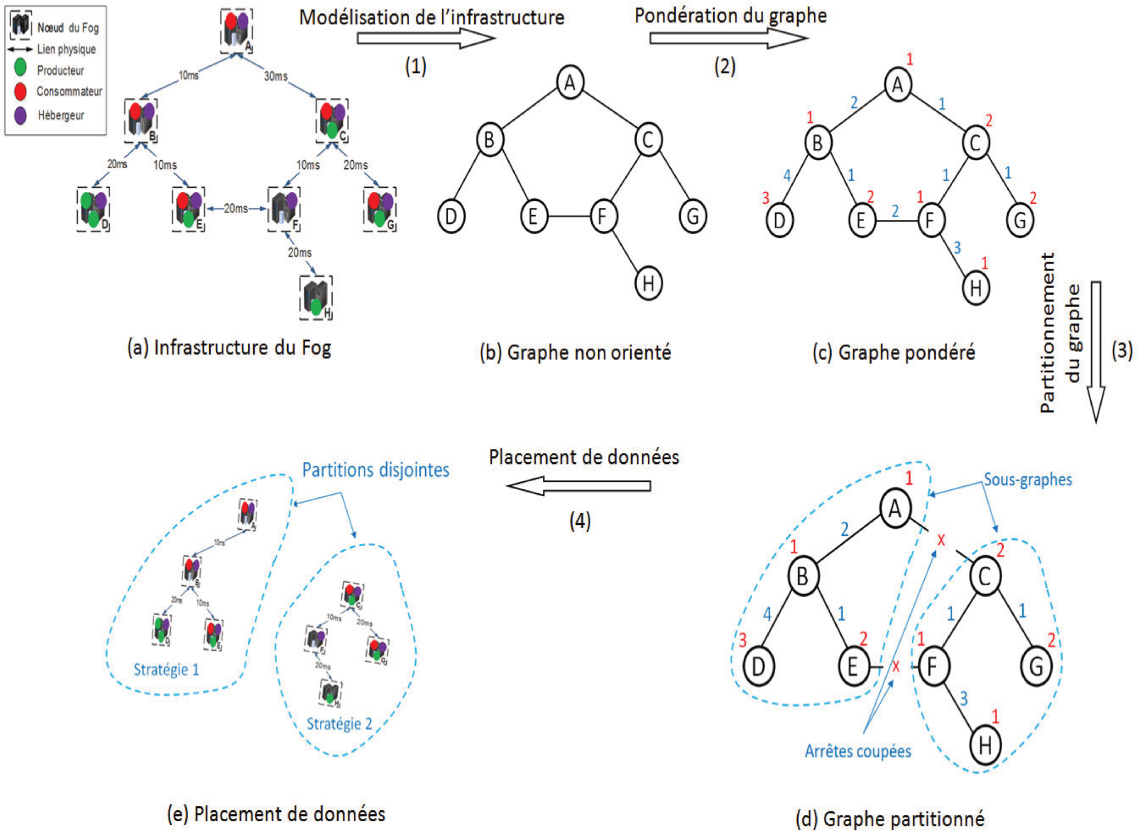


FIGURE 11 : Les différentes étapes de l'heuristique *iFogStorG*.

Dans les sections suivantes, nous décrivons chacune des étapes mentionnées précédemment.

5.3 MODÉLISATION DE L'INFRASTRUCTURE SOUS FORME DE GRAPHE

Dans cette section, nous allons, tout d'abord, décrire le modèle du système utilisé. Par la suite, nous modélisons l'infrastructure du système sous forme de graphe.

5.3.1 *Modélisation du système*

Considérons le système S composé d'un ensemble de nœuds de *Fog* noté $FN = \{fn_1, fn_2, ..., fn_n\}$, un ensemble d'instances de services d'IoT considérées comme des producteurs de données noté $DP = \{dp_1, dp_2, ..., dp_l\}$, et un ensemble d'instances de services d'IoT considérées comme des consommateurs de données noté $DC = \{dc_1, dc_2, ..., dc_m\}$. Soit $DH = \{dh_1, dh_2, ..., dh_k\}, DH \in FN$, le sous-ensemble de nœuds de *Fog* considérés comme des hébergeurs de données.

Soit DFM la matrice des flux de données qui relie chaque producteur avec l'ensemble des consommateurs associés. Le coefficient $f_{i,j} = 1$ signifie que le consommateur dc_j utilise les données générées par le producteur dp_i.

$$DFM = \begin{bmatrix} f_{1,1} & \cdots & f_{1,m} \\ \vdots & \ddots & \vdots \\ f_{l,1} & \cdots & f_{l,m} \end{bmatrix}, f_{i,j} \in \{0,1\} \tag{14}$$

Soit DPM la matrice des emplacements des producteurs de données. Le coefficient $p_{g,i} = 1$ signifie que le nœud fn_g héberge le producteur dp_i, sinon $p_{g,i} = 0$.

$$DPM = \begin{bmatrix} p_{1,1} & \cdots & p_{1,l} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,l} \end{bmatrix}, p_{g,i} \in \{0,1\} \tag{15}$$

Soit DCM la matrice des emplacements des consommateurs de données. Le coefficient $c_{g,j} = 1$ signifie que le nœud fn_g héberge le consommateur dc_j. Sinon $c_{g,j} = 0$.

$$DCM = \begin{bmatrix} c_{1,1} & \cdots & c_{1,m} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,m} \end{bmatrix}, c_{g,j} \in \{0,1\} \tag{16}$$

ADM est la matrice d'adjacence des nœuds de *Fog*. Le coefficient $t_{g,u}$ représente la latence réseau (en millisecondes) du lien physique existant entre le nœud fn_g et le nœud fn_u. Si $t_{g,u} = \infty$ cela signifie qu'il n'y a aucun lien physique direct existant entre fn_g et fn_u.

$$ADM = \begin{bmatrix} t_{1,1} & \cdots & t_{1,n} \\ \vdots & \ddots & \vdots \\ t_{n,1} & \cdots & t_{n,n} \end{bmatrix}, t_{g,u} \in \mathfrak{R}^{+} \tag{17}$$

5.3.2 *Construction du graphe*

Comme illustré dans la Figure 11 (a) et (b), l'infrastructure du système S (défini dans la section précédente) est modélisée sous forme de graphe non-orienté $G = (V, E)$, tel que, $V = \{v_1, ..., v_n\}$ est l'ensemble des nœuds du graphe qui représentent les nœuds de *Fog*, et $E = \{e_1, ..., e_r\}$ est l'ensemble des arêtes du graphe qui représentent les liens physiques du réseau dans l'infrastructure. Le graphe G est pondéré en nœuds et en arêtes dans l'étape suivante.

5.4 PONDÉRATION DU GRAPHE

La pondération du graphe est réalisée en deux étapes ; (i) pondération des nœuds, et (ii) pondération des arêtes. Nous décrivons chaque étape ci-après.

5.4.1 *Pondération des nœuds*

Nous rappelons que les poids des nœuds du graphe sont utilisés pour équilibrer la complexité des sous-problèmes de placement de données dans les différentes partitions de l'infrastructure. La complexité d'un problème de placement de données dépend de la quantité de données à placer et du nombre de nœuds de *Fog* existant dans l'infrastructure. Afin de répartir la quantité de données et les nœuds de *Fog* entre les différentes parties de l'infrastructure, notre idée consiste à pondérer chaque nœud du graphe par le nombre de producteurs de données déployés dans ce nœud. De plus, le poids d'un nœud sera incrémenté par 1 si le nœud est aussi un hébergeur de données. Cela est fait afin de répartir les nœuds de *Fog* équitablement entre les différents sous-graphes.

Par exemple, dans la Figure 11 (a), le nœud de *Fog* D déploie deux producteurs de données, et il est considéré comme hébergeur de données. Le poids affecté à ce nœud est donc de 3, voir la Figure 11 (c). L'Algorithme 1 décrit la pondération des nœuds du graphe avec :

- calculNombreDPdansFN : la fonction qui retourne le nombre de producteurs hébergés dans un nœud de *Fog*.
- PoidsNœuds : un vecteur qui contient tous les poids des nœuds du graphe.

Algorithme 1 : Pondération des nœuds du graphe.

Entrées : G, DPM
Output : PoidsNœuds

```
1 function CalculPoidsNœuds()
2     pour v ∈ V(G) faire
3         PoidsNœuds[v] ← calculNombreDPdansFN(v, DPM)
4         si v est un dh alors
5             PoidsNœuds[v] ← PoidsNœud[v] + 1
6     retourner PoidsNœuds      // retourner le vecteur des poids des
      noeuds
```

5.4.2 *Pondération des arêtes*

Comme mentionné au début de ce chapitre, les flux de données inter-partition provoquent une dégradation de la qualité de la solution produite, et donc un mauvais placement de données. La génération des flux se passe lors de la connexion des arêtes (i.e. les liens physiques) lors de la subdivision de l'infrastructure. En effet, la déconnexion d'une arête rend tous les flux de données empruntant cette arête à des flux de données inter-partition. Afin de diminuer le nombre de flux de données inter-partition produits par le partitionnement de l'infrastructure, notre idée consiste à pondérer chaque arête du graphe par le nombre de flux de données passant par cette arête. Par conséquent, la minimisation de la somme des poids des arêtes déconnectées, lors du partitionnement, minimise le nombre de flux de données inter-partition produits.

Afin de compter le nombre de flux de données passant par un lien physique de réseau, nous supposons que les échanges de données passent toujours par les plus courts chemins (en terme de latence) existants entre les producteurs, les hébergeurs et les consommateurs de données (voir la Section 2.5 du chapitre 2). En effet, la transmission de données à travers d'autres chemins augmente la latence du service, ce qui est en contradiction avec notre objectif initial qui consiste à minimiser la latence du service. Nous utilisons l'algorithme de *Floyd Warshall* [53] pour calculer tous les plus courts chemins existants entre les différents nœuds de l'infrastructure.

Par exemple, nous supposons que le producteur de données hébergé dans le nœud H de la Figure 11 (b) a trois flux de données : $(f_1 = (H \rightarrow B), f_2 = (H \rightarrow$

$C), f_3 = (H \rightarrow E)$). Le lien physique (H, F) est donc utilisé comme une partie du plus court chemin par ces trois flux de données. Ainsi, le poids alloué à l'arête associée à ce lien réseau est de 3.

L'Algorithme 2 décrit la pondération des arêtes avec :

- PlusCourtChemin : la fonction qui retourne le plus court chemin existant entre deux nœuds de *Fog* (i.e. le nœud de départ et le nœud d'arrivé d'un flux de données *f*).
- Chemin : un plus court chemin.
- PoidsArêtes : un tableau qui stocke tous les poids des arêtes du graphe.

Algorithme 2 : Pondération des arêtes du graphe.

Entrées : G, DFM,DPM,DCM,ADM
Output : PoidsArêtes

```
1 function CalculPoidsArêtes()
2     pour e ∈ E(G) faire
3         PoidsArêtes[e] ← 0                          // initialisation
4         pour f ∈ DFM faire
5             Chemin ← PlusCourtChemin(f, DPM, DCM, ADM)
6             si e ∈ Chemin alors
7                 PoidsArêtes[e] ← PoidsArêtes[e] + 1
8     retourner PoidsArêtes      // retourner le vecteur des poids des
       arêtes
```

5.5 PARTITIONNEMENT DU GRAPHE

Une fois le graphe pondéré, il est partitionné en K sous-graphes. L'objectif du processus de partitionnement consiste à minimiser la somme des poids des arêtes déconnectées en respectant la contrainte suivante : les poids des nœuds doivent être répartis équitablement entre les sous-graphes. Nous avons formulé ce problème comme suit.

Soient $P = \{P_1, ..., P_k\}$ un ensemble de K sous-graphes et $C = \{e_1, ..., e_w\}$, $C \subset E$, l'ensemble des arêtes déconnectées par le partitionnement du graphe G.

Considérons $Poids(P_z)$ et $Poids(C)$ comme étant les deux fonctions qui retournent respectivement la somme des poids des nœuds de P_z et la somme des poids des arêtes de C.

La formulation du problème de partitionnement du graphe est décrite ci-dessous [73] :

$$\begin{cases} \text{Minimiser} & \text{Poids}(C) \\ \text{Sous contraintes.} & \\ & \text{Poids}(P_1) \cong \ldots \cong \text{Poids}(P_k) \\ & \bigcup_{1 \leqslant i \leqslant k} P_i = V \\ & P_i \cap P_j = \emptyset, \forall i, j \in [1, k], i \neq j \end{cases}$$

Avec $a \cong b$ signifie que $a = b + \epsilon$, $\epsilon \in \mathfrak{R}^+$.

5.6 PLACEMENT DE DONNÉES

Dans cette étape, l'heuristique *iFogStorG* subdivise l'infrastructure du système en K partitions selon le résultat du partitionnement du graphe, voir la Figure 11 (e). Pour chaque partition, un nouveau sous-problème de placement de données est formulé et résolu indépendamment des autres en utilisant la stratégie de placement de données *iFogStor* [76]. Ensuite, les solutions des sous-problèmes sont agrégées afin de construire la solution globale de placement de données. L'approche par partitionnement est indépendante de la méthode de résolution des sous-problèmes. En effet, d'autres stratégies qu'*iFogStor* peuvent être utilisées pour formuler et résoudre les sous-problèmes de placement de données.

L'augmentation du nombre de partitions K aide à réduire la taille des sous-problèmes de placement de données, et par conséquent, elle réduit le temps de placement de données. D'autre part, cette augmentation génère des flux de données inter-partition supplémentaires (i.e. par la déconnexion d'autres arêtes), ce qui dégrade potentiellement la qualité de placement de données. Le compromis entre le gain en temps de placement et la perte de qualité de placement engendrés par l'augmentation du nombre de partitions est étudié dans la partie expérimentale de ce chapitre.

5.7 ÉVALUATION ET RÉSULTATS

Dans cette partie, nous commençons d'abord par présenter le scénario et les différents outils et configurations utilisés dans nos expérimentations. Ensuite, nous détaillons notre méthodologie pour évaluer l'heuristique *iFogStorG*. Nous finissons par présenter et discuter des résultats d'expérimentations.

5.7.1 *Scénario d'évaluation*

Pour évaluer l'heuristique *iFogStorG*, nous avons utilisé le scénario d'évaluation défini dans la Section 4.6.1 du chapitre 4. Nous rappelons que dans ce scénario,

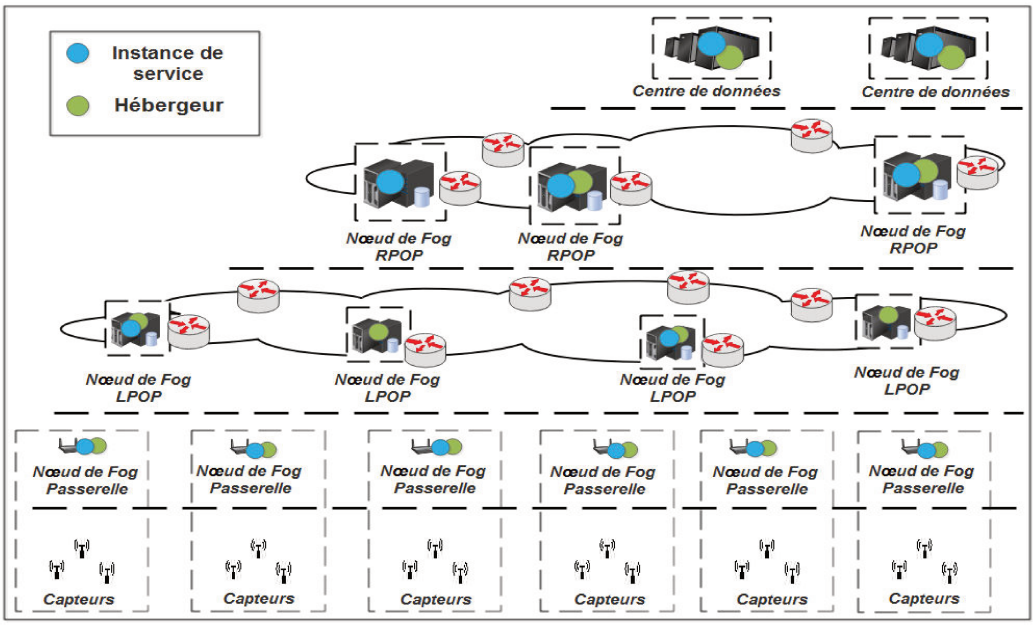


FIGURE 12 : L'infrastructure du scénario d'évaluation.

un ensemble de capteurs surveillent l'environnement et renvoient des informations à un ensemble d'instances de services. Les instances de services sont déployées dans une infrastructure qui comprend des nœuds de *Fog* et des centres de données dans le *Cloud*. Comme illustré dans la Figure 12, l'infrastructure du scénario comprend des passerelles, LPOP, RPOP et centres de données. Dans ce scénario, les instances de services peuvent (i) demander des données de sources différentes, et (ii) partager des données entre elles. Nous avons utilisé trois répartitions de données différentes dans nos expérimentations décrites comme suit :

- **Répartition zonée :** ici, les données sont utilisées par des consommateurs localisés dans les mêmes partitions géographiques que les producteurs. Dans nos expérimentations, une partition est définie par l'espace géographique couvert par un RPOP.
- **Répartition distribuée :** dans ce cas, les données sont utilisées par des consommateurs distribués sur l'ensemble de l'infrastructure.
- **Répartition mixte :** ici, la moitié des données est répartie suivant une répartition *zonée* et l'autre moitié suivant une répartition *distribuée*.

5.7.2 *Outil et configurations*

Afin d'évaluer *iFogStorG*, nous avons utilisé :

- *iFogSim* [62] pour simuler le comportement du système, déployer nos stratégies de placement de données et mesurer la latence globale générée dans le système.

- *Metis* [9] pour partitionner le graphe en K sous-graphes.
- *iFogStor* pour placer les données dans les différentes partitions de l'infrastructure.

Comme illustré dans la Figure 13, tout d'abord, le simulateur *iFogSim* démarre et génère l'infrastructure du système (i.e. création des capteurs, nœuds de *Fog*, centres de données, instances de services et flux de données). Ensuite, ce simulateur modélise l'infrastructure générée comme un graphe pondéré au niveau des nœuds et des arêtes. Une fois le graphe pondéré, *iFogSim* passe ce graphe à l'outil *Metis*. Puis, ce dernier partitionne le graphe en K sous-graphes en appelant la méthode de *K-way partitioning*, et retourne le résultat à *iFogSim*. Ensuite, ce dernier subdivise l'infrastructure du système selon le résultat du partitionnement du graphe. Ensuite, pour chaque partie, *iFogSim* fait appel à *iFogStor* pour formuler et résoudre le problème de placement de données dans cette partie. Une fois les sous-problèmes résolus, *iFogSim* configure les emplacements de données et commence la simulation. Il est à noter qu'il est possible d'utiliser d'autres stratégies qu'*iFogStor* pour placer les données dans les sous-parties de l'infrastructure.

L'infrastructure simulée dans ce travail comporte 5 centres de données, 10 RPOP, 100 LPOP et 10.000 passerelles (voir la Section 4.6.2 du chapitre 4). Nous avons fixé 100 capteurs par passerelle. Nous rappelons que les latences réseau existantes entre les différents nœuds de *Fog*, et la capacité de stockage libre existante dans chaque hébergeur de données sont décrites respectivement dans la Table 3 et la Table 4 du chapitre 4.

De manière similaire aux expérimentations du chapitre 4, nous avons considéré que les données sont échangées à base de paquets. Nous avons fixé la taille d'une donnée générée par un capteur à 96 octets et celle générée par une instance de service à 960 octets [37].

Nous avons fait varier le nombre d'instances de services d'IoT qui partagent les mêmes données (noté CP) de 3 à 5, et pour chaque cas, nous avons étudié la latence globale générée dans le système.

Les expérimentations ont été réalisées en utilisant une machine avec un CPU 48 cœurs *Xeon E5-2650* cadencés à 2.2 GHz et de 348 Go de RAM, en utilisant la distribution *Ubuntu* version 16.04 du système d'exploitation GNU/*Linux*.

5.7.3 *Méthodologie d'évaluation*

Afin d'évaluer l'heuristique de placement de données *iFogStorG*, nous avons considéré deux métriques : (i) la qualité de placement de données qui est mesurée par la somme de toutes les latences générées par la transmission de données sur une période de temps, et (ii) le temps nécessaire pour calculer le placement

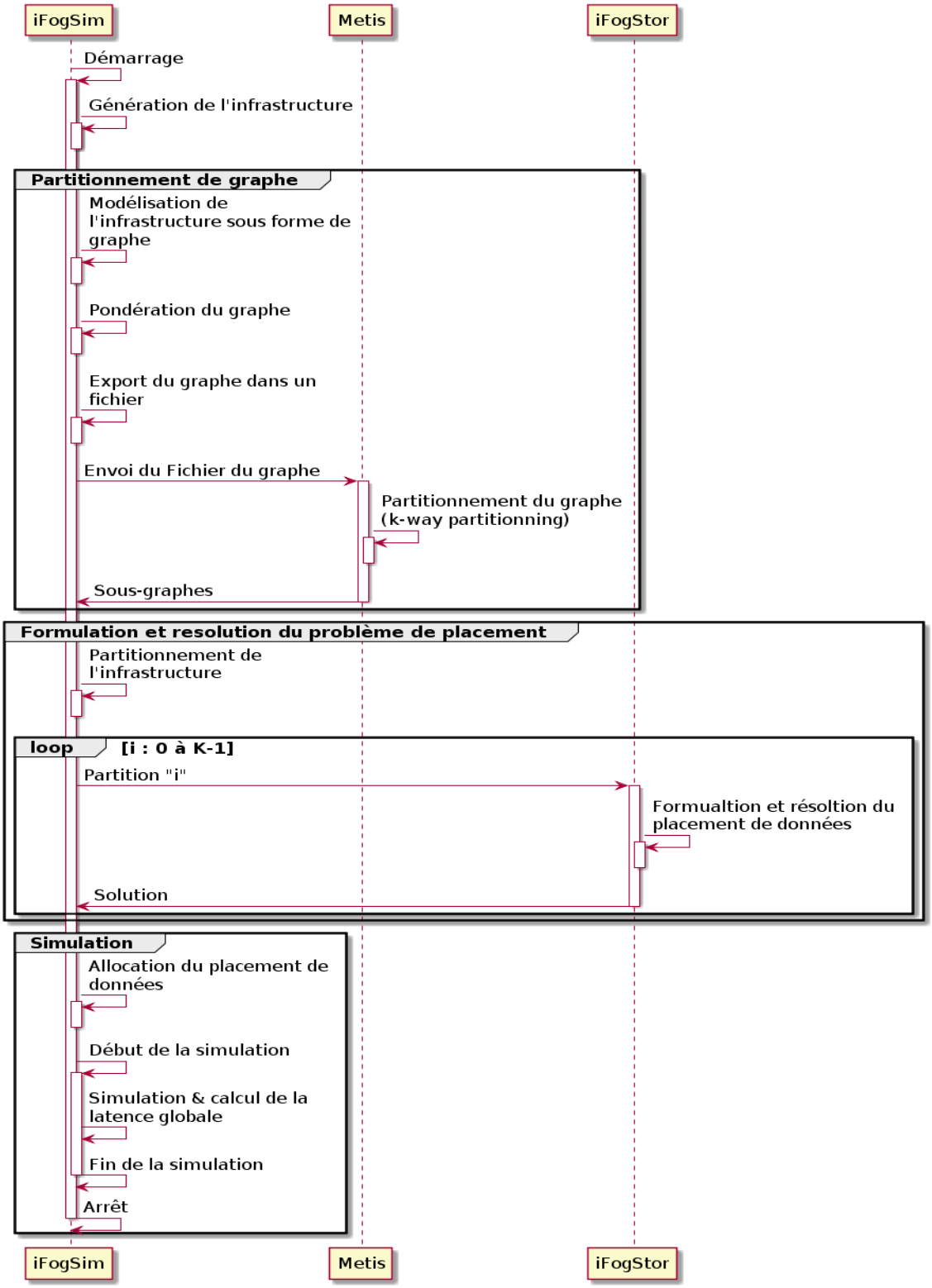


FIGURE 13 : Le diagramme de séquence de la simulation.

de données (dénoté $place_{time}$). Ce dernier est égal à la somme de temps du partitionnement de l'infrastructure (dénoté $parti_{time}$), de temps de la formulation (dénoté $formu_{time}$) des sous-problèmes et de temps de leur résolution (dénoté $solvi_{time}$), voir l'équation 18. Nous comparons ces deux métriques pour les stratégies de placement de données suivantes :

$$place_{time} = parti_{time} + formu_{time} + solvi_{time} \tag{18}$$

- *iFogStor :* c'est la stratégie exacte proposé dans la Section 4.5.1 du chapitre 4 pour placer les données. Elle trouve la solution optimale pour placer les données en minimisant la latence globale du système.
- *iFogStorZ :* c'est la première heuristique proposée dans la Section 4.5.2 du chapitre 4 pour placer les données. Elle subdivise l'infrastructure géographiquement en plusieurs partitions. Pour chaque partition, elle fait un appel à *iFogStor* pour placer les données.
- *iFogStorG :* c'est la deuxième heuristique (objet de ce chapitre) pour placer les données. Elle subdivise l'infrastructure en K sous partitions en utilisant des méthodes basées sur la théorie des graphes. Pour chaque partition, un sous-problème de placement de données est modélisé et résolu en utilisant *iFogStor*.

Afin de comparer les deux heuristiques, nous avons utilisé le même nombre de partitions $K = 2$, 5 et 10 utilisé dans le chapitre 4. Pour tester le passage à l'échelle d'*iFogStorG*, qui est évalué par la qualité de placement en considérant un grand nombre de partitions, nous avons construit des scénarios avec 20, 50 et 100 partitions. Ces configurations ne sont pas testées avec *iFogStorZ*, car le nombre de partitions géographiques existantes dans notre scénario d'évaluation se limite à 10.

5.7.4 *Résultats et discussion*

Dans cette partie, nous présentons d'abord les résultats obtenus concernant la qualité de placement de données qui est mesurée par la latence globale générée dans le système. Ensuite, nous décrivons des résultats obtenus pour le temps d'exécution du processus de placement des données.

5.7.4.1 *Qualité de placement de données*

Les Figures 14, 15 et 16 montrent les latences globales générées dans le système en faisant varier le paramètre CP de 3 à 5. Nous observons que l'augmentation des latences suit la même tendance que l'augmentation de CP. Cela est dû à

la charge supplémentaire introduite par le transfert de données aux différents consommateurs.

La Table 7 illustre la qualité de placement de chaque heuristique en faisant varier le nombre de partitions. La qualité de placement est calculée par la division de la valeur de la latence générée par l'algorithme exact *iFogStor* sur la valeur de la latence générée par l'heuristique pour un nombre de partitions donné (i.e. 100% représente l'optimum).

Tout d'abord, en utilisant une répartition de données de type *zonée* qui est supportée par *iFogStorZ*, les deux heuristiques *iFogStorZ* et *iFogStorG* ont approximativement les mêmes performances que l'algorithme exact *iFogStor*. Cela est vrai pour les différents nombres de partitions utilisés. Comme illustré dans la Table 7, la qualité de placement de données est proche de l'optimal pour les deux heuristiques. D'autre part, *iFogStorG* s'adapte bien à l'augmentation du nombre de partitions. Dans le pire des cas évalués (i.e. 100 partitions), *iFogStor* assure plus de 92% d'optimalité de placement de données.

Deuxièmement, en utilisant une répartition de données de type *mixte*, nous observons que les latences augmentent pour toutes les stratégies. Cela est dû au temps de transfert de données supplémentaire dû à la transmission de données aux consommateurs localisés dans des partitions autres que les producteurs. Nous observons aussi que, par l'augmentation du nombre de partitions, *iFogStorZ* génère plus de latence et perd significativement en qualité de placement. Par exemple, la qualité de placement passe de 95% en utilisant 2 partitions à 80% en utilisant 10 partitions. À l'inverse, *iFogStorG* est peu influencé par l'augmentation du nombre de partitions et garde une qualité de placement proche de l'optimale (97%). De plus, cette heuristique basée sur la théorie des graphes, passe l'échelle et garde la qualité de placement au dessus de 97% avec l'utilisation de 100 partitions pour une répartition de type *mixte*.

Enfin, en utilisant une répartition de données de type *distribuée*, nous observons que les latences sont proportionnelles au nombre de partitions. *iFogStorZ* génère des latences élevées et perd significativement en qualité de placement. Comme illustré dans la Table 7, en utilisant *iFogStorZ*, la qualité de placement de données est diminuée à 92% en utilisant 2 partitions et à 69% en utilisant 10 partitions. Tandis qu'*iFogStorG* garde de bonnes performances pour les différents nombres de partitions. Comme illustré dans la Table 7, *iFogStorG* a une optimalité de 95% en utilisant 10 partitions et de 91% pour 100 partitions.

Pour résumer, *iFogStorG* présente une bonne qualité de placement et garde de bonnes performances quel que soit le type de la répartition de données utilisée. En revanche, *iFogStorZ* se comporte moins bien quand les données sont utilisées par des consommateurs localisés dans des partitions différentes de celles des producteurs (i.e. répartitions de type *mixte* et *distribuée*). D'autre part, *iFogStorG* peut utiliser plus de 100 partitions en gardant une qualité de placement élevée.

Table 7 : Qualité de placement des heuristiques.

Stratégie	Zonée	Mixte	Distribuée
iFogStorZ_2	99.99%	95.72%	92.38%
iFogStorZ_5	99.99%	85.72%	76.31%
iFogStorZ_10	99.99%	80.82%	69.72%
iFogStorG_2	99.97%	97.64%	96.12%
iFogStorG_5	99.99%	97.17%	95.49%
iFogStorG_10	99.99%	97.30%	95.49%
iFogStorG_20	99.22%	97.29%	95.50%
iFogStorG_50	99.92%	97.43%	95.50%
iFogStorG_100	92.83%	97.54%	91.38%

Cela montre la capacité d'*iFogStorG* de passer l'échelle en termes de nombre de partitions. *iFogStorZ*, à l'inverse, est limitée par le nombre de partitions géographiques. Dans la section suivante, nous allons voir que le passage à l'échelle permet des gains importants en temps de calcul pour le placement.

5.7.4.2 *Temps de calcul de placement de données*

Dans cette partie, nous comparons le temps de calcul de la solution de placement de données pour chaque stratégie. Comme mentionné auparavant, le temps de calcul de placement est égal à la somme de temps du partitionnement de l'infrastructure, de temps de la formulation des sous-problèmes et de temps de leur résolution (voir l'équation 18).

La Figure 17 montre les temps d'exécution des différentes phases de chaque stratégie de placement. Notons que la nature de la répartition des données n'a pas d'influence sur le temps d'exécution de la stratégie. En effet, la taille des sous-problèmes reste approximativement la même pour les différentes répartitions de données utilisées. Dans la Figure 17, nous observons que le temps de formulation et le temps de résolution du problème sont très élevés en utilisant l'algorithme exact *iFogStor*. Cela reflète la nature du problème de placement de données qui est *NP-difficile*. D'autre part, l'augmentation du nombre de partitions réduit significativement ces temps. Cela est dû à la réduction de la complexité du problème par le découpage de l'infrastructure. Nous observons aussi que le temps de partitionnement est négligeable (moins d'une seconde) en comparaison avec le temps de la formulation et le temps de résolution du problème de placement de données. Notons qu'*iFogStorZ* utilise un découpage géographique de l'infrastructure, le temps de partitionnement de cette stratégie est donc quasi nul.

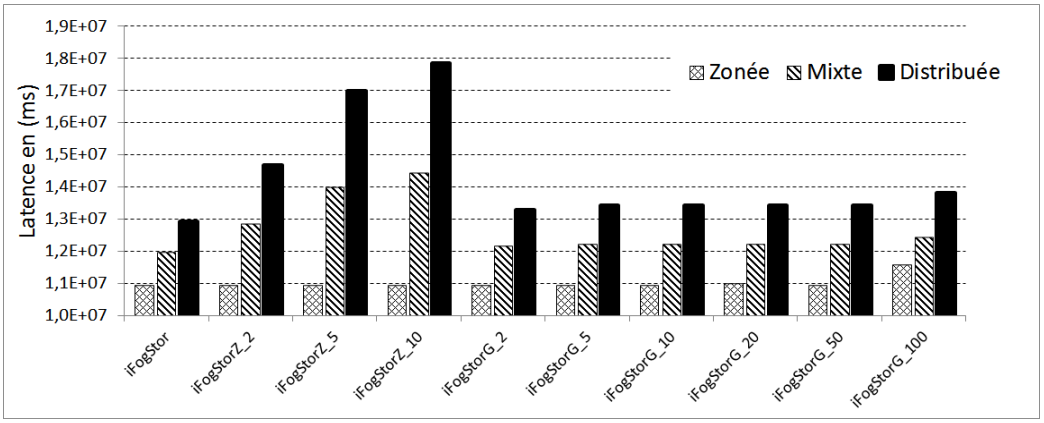


FIGURE 14 : La latence globale du système pour CP = 3.

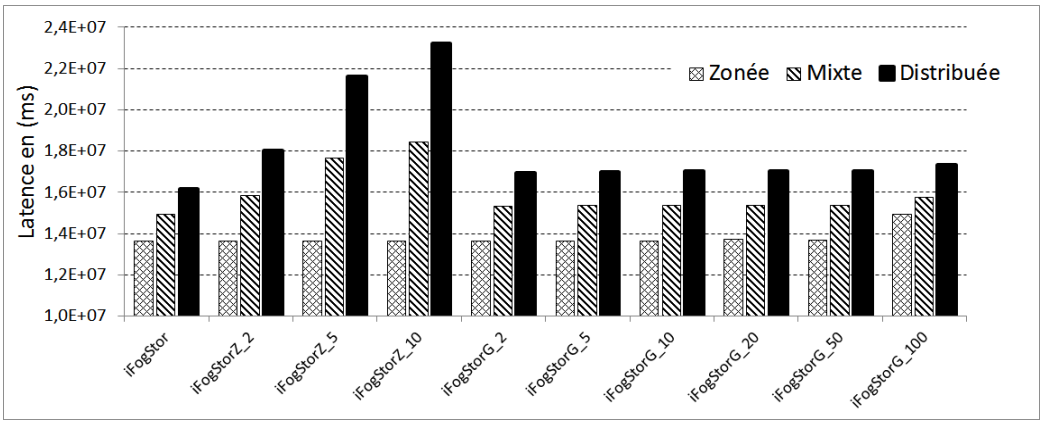


FIGURE 15 : La latence globale du système pour CP = 4.

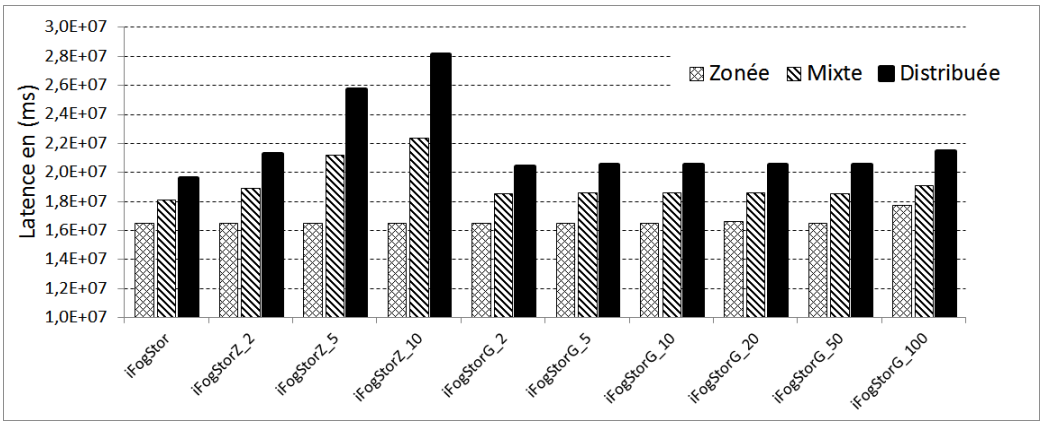


FIGURE 16 : La latence globale du système pour CP = 5.

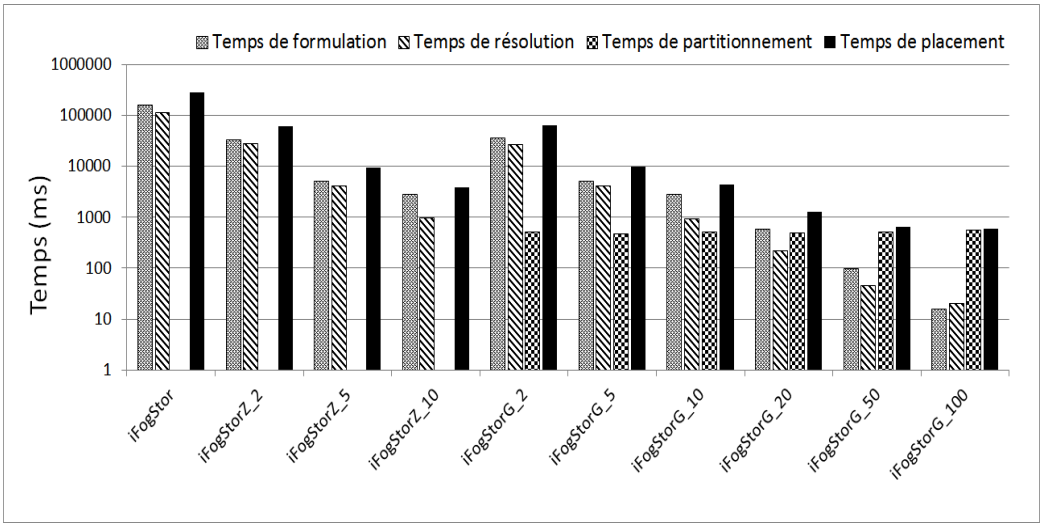


FIGURE 17 : Temps de placement de données.

TABLE 8 : Le gain en temps de placement des heuristiques

Stratégie	iFogStorZ_2	iFogStorZ_5	iFogStorZ_10
Gain	4.83	35.46	81.22
Stratégie	iFogStorG_2	iFogStorG_5	iFogStorG_10
Gain	5.09	32.96	90.26
Gain	iFogStorG_20	iFogStorG_50	iFogStorG_100
Gain	223.17	442.70	473.55

La Table 8 illustre le gain en temps de placement pour chaque heuristique par l'augmentation du nombre de partitions. Ce gain est obtenu par la division du temps de placement d'*iFogStor* par le temps de placement pris par chaque heuristique. Nous observons que les deux heuristiques présentent approximativement les mêmes performances. Par exemple, *iFogStorZ* et *iFogStorG* réduisent le temps de placement de 5 fois en utilisant 2 partitions, et de 30 fois en utilisant 5 partitions. *iFogStorG* peut réduire le temps de placement par plus de 473 fois en utilisant 100 partitions. Notons que le nombre de partitions d'*iFogStorG* peut varier entre 1 et le nombre de nœuds de *Fog* existants dans l'infrastructure.

Pour résumer, le temps de placement de données est très élevé dans les grandes infrastructures en utilisant l'algorithme exact *iFogStor*. Cependant, ce temps peut être réduit significativement par l'augmentation du nombre de partitions en utilisant des heuristiques basées sur l'approche "diviser pour régner", permettant ainsi un placement de données en ligne.

5.8 CONCLUSION

Dans ce chapitre, nous avons présenté *iFogStorG*, une heuristique pour placer en ligne les données dans les infrastructures de *Fog*. Nous avons suivi une méthode basée sur le concept "diviser pour régner"afin de réduire la complexité du problème de placement en le subdivisant en plusieurs sous-problèmes solvables séparément et éventuellement en parallèle. Afin de minimiser la perte d'information lors du processus de subdivision, qui peut dégrader la qualité de la solution, nous avons utilisé une méthode basée sur la théorie des graphes. Nous avons commencé par modéliser l'infrastructure du système comme un graphe non-orienté qui est par la suite pondéré en nœuds et en arêtes. Les nœuds du graphe représentent les nœuds de *Fog*, et les arêtes modélisent les liens physiques du réseau. Les poids des nœuds sont utilisés pour équilibrer la complexité entre les sous-graphes, et les poids des arêtes sont utilisés pour minimiser la perte d'information. Une fois le graphe pondéré, nous avons utilisé l'outil *Metis* pour le partitionner en K sous-graphes définissant K sous-problèmes de placement avec des complexités réduites. Enfin, nous appelons *iFogStor* pour placer les données pour chaque partie de l'infrastructure.

Nous avons évalué *iFogStorG* en utilisant un scénario générique de "ville intelligente". Les expérimentations montrent qu'*iFogStorG* a une bonne qualité de placement de données et garde de bonnes performances en augmentant le nombre de partitions (pour avoir plus d'accélération en temps de placement). De plus, cette heuristique présente plus de flexibilité.

Dans le chapitre suivant, nous allons présenter notre troisième contribution concernant la gestion de placement de données dans les infrastructures de *Fog*. Cette contribution a pour but de pallier le manque d'outils d'évaluation des stratégies de placement de données dans le contexte de *Fog*. Pour cela, nous avons réalisé une extension du simulateur *iFogSim* afin de pouvoir déployer, tester et comparer des stratégies d'optimisation de placement de données de l'IoT dans le *Fog*.

Chapitre 6

UNE EXTENSION D'IFOGSIM POUR ÉVALUER DES STRATÉGIES DE PLACEMENT DE DONNÉES

Dans ce chapitre, nous proposons une extension d'*iFogSim*, un simulateur d'environnements de *Fog* et d'IoT. L'objectif de cette extension consiste à produire une plate-forme dédiée à implanter, tester et évaluer des stratégies de placement de données dans un environnement de *Fog* et d'IoT. Ce chapitre présente dans un premier lieu le simulateur *iFogSim* puis un aperçu de l'extension proposée. Ensuite, nous décrivons les différents composants et fonctionnalités rajoutés à *iFogSim*. Nous finissons par discuter des résultats de test et d'évaluation de l'extension proposée.

Sommaire

6.1 LE SIMULATEUR *ifogsim*

iFogSim est un simulateur d'environnements de *Fog* et d'IoT [62]. C'est une extension du simulateur d'environnements de *Cloud* : *CloudSim* [35]. *iFogSim* est conçu pour pouvoir évaluer des stratégies de gestion des ressources applicables aux environnements de *Fog* et d'IoT en ce qui concerne leur impact sur la latence, la consommation d'énergie, la congestion du réseau et les coûts opérationnels.

iFogSim permet de simuler un environnement de *Fog* et d'IoT incluant des capteurs, des actionneurs, des nœuds de *Fog* et des centres de données de *Cloud*. Les applications d'IoT simulées dans *iFogSim* sont basées sur le modèle *Perception-Traitement-Action* [62]. Dans ce modèle, un ensemble de capteurs (ex. de température) collectent d'abord des informations relatives à l'environnement physique de l'application, puis les publient de manière périodique ou événementielle (ex. si la température dépasse un certain seuil). Ensuite, ces informations sont récupérées puis traitées par un ensemble d'instances de services d'IoT déployées dans le *Fog* et dans le *Cloud*. Enfin, conformément aux résultats du traitement, des messages sont envoyés aux actionneurs pour agir sur l'environnement physique de l'application (ex. éteindre le chauffage ou déclencher une alarme).

Afin de réaliser une simulation dans *iFogSim*, les utilisateurs spécifient l'infrastructure physique de système (i.e. capteurs, actionneurs, nœuds de *Fog*, liens réseaux, etc.), le scénario (i.e. services d'IoT, flux de données, etc.), leurs stratégies de placement et/ou d'ordonnancement de services, et le temps maximal de la simulation. *iFogSim* produit en sortie la valeur d'un ou de plusieurs critères (selon les objectifs des utilisateurs) parmi ceux qui sont mentionnés auparavant.

La Figure 18 montre l'architecture initiale d'*iFogSim* (i.e. sans l'extension proposée). Ce simulateur est composé : (i) d'un ensemble d'entités permettant de modéliser les équipements physiques de l'infrastructure, (ii) un ensemble d'entités permettant de modéliser les éléments logiques du scénario simulé (ex. les données et les instances des services de l'IoT), et (iii) un ensemble d'entités permettant de gérer les ressources de traitement dans les nœuds de *Fog* et dans les centres de données. Ces ressources sont manipulées pour placer et ordonnancer les instances de services dans les entités physiques afin d'optimiser la latence du service, le trafic réseau, la consommation énergétique ou le coût opérationnel du système. Ci-après nous décrivons chaque ensemble d'entités définies dans *iFogSim*.

- **Les entités physiques :** ce sont les modèles des équipements physiques trouvés dans une infrastructure de *Fog* (ex. serveurs, capteurs, actionneurs etc.).
    1. **FogDevice :** cette entité modélise les nœuds de *Fog* (ex. les switchs, les routeurs, les stations de base, les passerelles, etc). Elle spécifie les

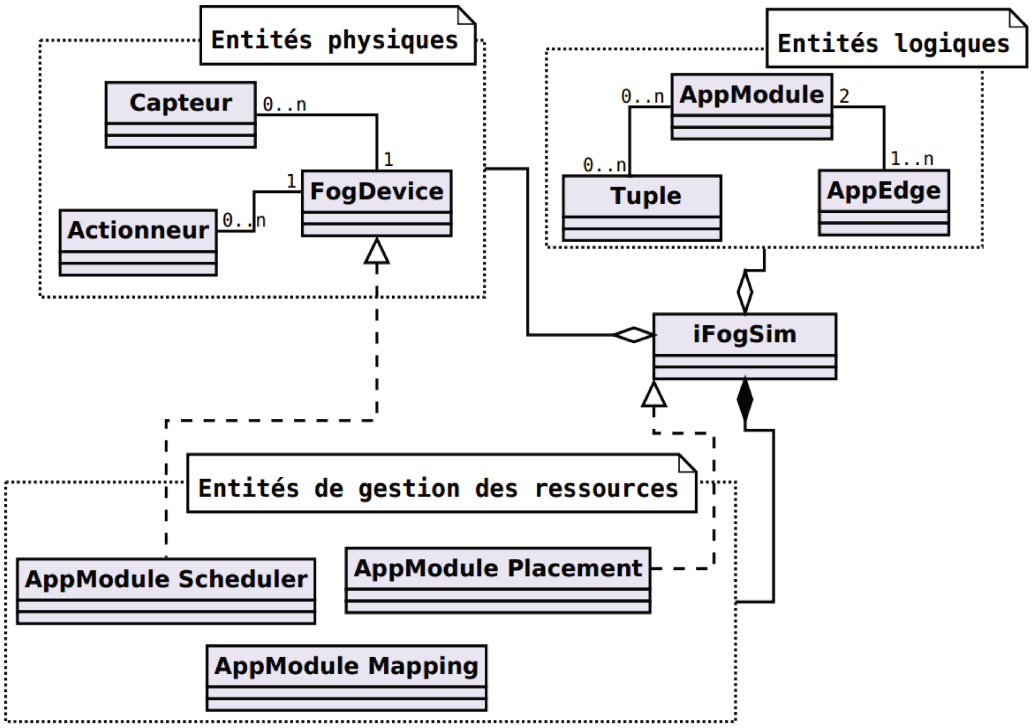


FIGURE 18 : L'architecture d'*iFogSim*.

caractéristiques matérielles d'un nœud de *Fog* comme le modèle du processeur, la taille de la RAM, la capacité du stockage et la bande passante du réseau.

2. **Capteur :** cette entité modélise les différents capteurs déployés dans l'infrastructure simulée. Elle spécifie les attributs d'un capteur allant de sa connectivité réseau jusqu'aux données capturées.
3. **Actionneur :** cette entité modélise les actionneurs et les effets de leur action sur l'environnement simulé. Elle spécifie les attributs d'un actionneur comme sa connectivité, sa latence et la passerelle associée.

- **Les entités logiques :** ce sont des modèles représentant les éléments logiques trouvés dans un environnement informatique (ex. données, services, traitements, etc.).
  1. **Tuple :** cette entité représente l'unité fondamentale des communications entre les entités physiques (i.e. les données et leurs métadonnées) dans *iFogSim*. Un *Tuple* est caractérisé par son type (ex. température), le nœud source, le nœud destinataire, la capacité du traitement de données demandée (en millions d'instruction) et la taille des données encapsulées (en octets).
  2. **AppModule :** cette entité représente les unités de traitement du scénario simulé qui sont les instances de services de l'IoT. Ces instances de

services sont déployées dans les nœuds de *Fog* et les centres de données. Pour chaque donnée d'entrée (*Tuple*), une *AppModule* s'occupe de son traitement et génère des données de sortie qui sont envoyées par la suite à une ou à plusieurs *AppModules*. Le taux de production des données de sortie est exprimé par un modèle de sélection des données déployé dans le simulateur *iFogSim*. Par exemple, un taux de sélection de 10% consiste à générer une donnée (i.e. une *Tuple*) de sortie pour 10 données d'entrée.

3. **AppEdge :** cette entité représente les dépendances de données existantes entre deux *AppModules* (i.e. un producteur et un consommateur de données). De plus, une *AppEdge* spécifie le mode de production de données parmi les deux modes possibles : périodique ou événementiel. Le modèle de dépendance de données (i.e. les dépendances entre les producteurs et les consommateurs de données) du scénario simulé est représenté par un graphe dirigé acyclique [62] composé d'un ensemble d'*AppEdges* interconnectées entre elles. Les nœuds du graphe représentent les *AppModules* et les arêtes représentent les flux de données.

- **Les entités de gestion de ressources :** ce sont des modèles représentant les stratégies de gestion de ressources physiques et logiques dans *iFogSim* (ex. allocation, ordonnancement, migration, etc.).

  1. **AppModule Placement :** cette entité définit un ensemble de méthodes permettant de gérer le placement des *AppModules* dans l'infrastructure simulée. Ainsi, les utilisateurs utilisent ces méthodes pour définir leur stratégie de placement des *AppModules* afin d'optimiser un ou plusieurs critères parmi ceux qui sont cités au début de cette section. Une fois les *AppModules* placés dans les nœuds de *Fog*, ils sont ordonnancés suivant l'entité *AppModule Scheduler* qui est définie par la suite.
  2. **AppModule Mapping :** cette entité définit pour chaque instance de service d'IoT, le nœud de *Fog* ou le centre de données qui l'héberge. Cette entité est configurée suivant la politique implantée dans l'entité de gestion de ressources *AppModule Placement*.
  3. **AppModule Scheduler :** cette entité définit un ensemble de méthodes pour gérer l'ordonnancement des *AppModules* dans un nœud de *Fog* ou un centre de données. Les utilisateurs utilisent ces méthodes pour définir leur stratégie d'ordonnancement des *AppModules*. Le mode d'ordonnancement mis en œuvre par défaut dans *iFogSim* subdivise équitablement les ressources de traitement entre l'ensemble des *AppModules* hébergés dans une entité physique de l'infrastructure.

Dans *iFogSim*, les entités physiques communiquent entre elles en utilisant des événements prédéfinis. Par exemple, il existe un événement pour lancer une instance de service dans un nœud de *Fog*, un événement pour connecter un capteur à une passerelle, et un événement pour envoyer des données à une autre entité. Les utilisateurs peuvent enrichir le système par l'ajout d'événements pour des cas spécifiques.

6.2 APERÇU DE L'EXTENSION

Dans ce chapitre, nous présentons notre troisième contribution concernant la gestion de placement de données dans les infrastructure de type *Fog*. Cette contribution consiste à étendre le simulateur *iFogSim*. Elle fournit aux utilisateurs une plate-forme expérimentale pour mettre en œuvre et tester leurs stratégies de placement de données au regard de la latence du service, du trafic réseau, de la consommation énergétique ou du coût opérationnel du système.

Comme illustré dans la Figure 19, nous avons ajouté trois composants au simulateur *iFogSim*. Ces composants sont décrits comme suit :

- **Placement de données :** c'est le composant principal de cette extension. Il fournit aux utilisateurs un ensemble de fonctions pour calculer un placement de données en utilisant des méthodes basées sur la programmation linéaire. En outre, ce composant définit un ensemble de stratégies de placement accessibles via une interface unifiée. Par ailleurs, cet ensemble peut être enrichie par l'ajout d'autres stratégies définies par les utilisateurs eux-même. Les emplacements de stockage de données sont spécifiés dans le *catalogue des emplacements* (ou *dictionnaire* des emplacements) qui est configuré suivant la stratégie de placement de données utilisée.

- **Partitionnement de l'infrastructure :** ce composant offre aux utilisateurs un ensemble de méthodes pour subdiviser l'infrastructure en plusieurs partitions. Ensuite, pour chaque partition, les données sont placées indépendamment. Ce composant permet de réduire la complexité du système dans le but de rendre le placement de données faisable en ligne.

- **Répartition des flux de données :** notre extension comprend un scénario générique de "ville intelligente". Ce scénario inclut un ensemble de capteurs et un ensemble de services d'IoT qui sont déployés dans un ensemble de nœuds de *Fog* et centres de données. Comme la nature de répartition des flux de données a un impact sur la qualité de la stratégie de placement utilisée (voir la Section 5.7.4 du chapitre 5), nous avons ajouté le composant *Répartition des flux de données* à *iFogSim*. Ce composant génère différentes distributions de flux de données permettant d'évaluer des stratégies de placement en utilisant des répartitions de données différentes.

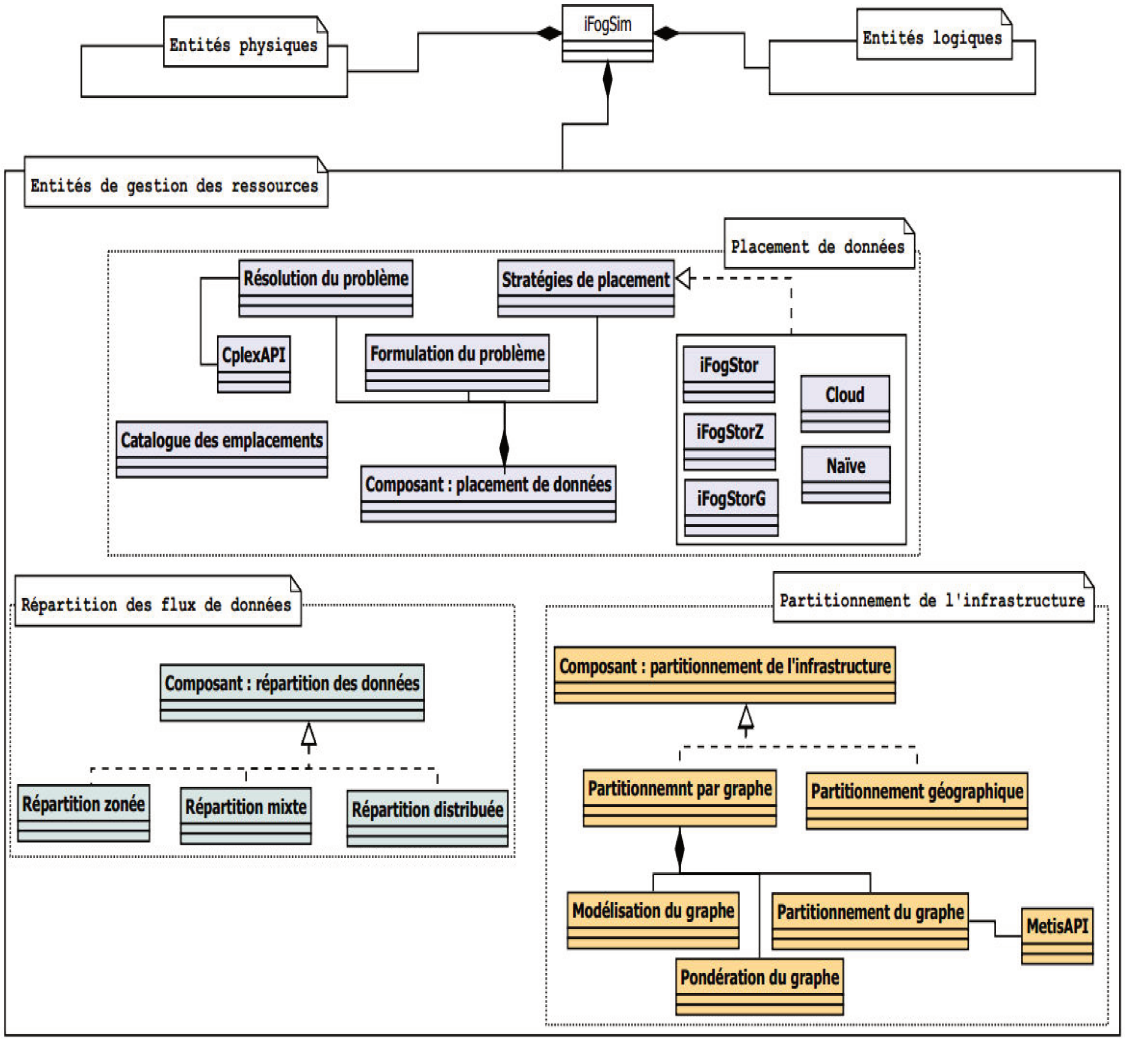


FIGURE 19 : Diagramme de classes UML des composants ajoutés à *iFogSim*.

Dans les sections suivantes, nous allons décrire chaque composant.

6.3 PLACEMENT DE DONNÉES

Ce composant offre un ensemble d'outils pour gérer le placement de données dans *iFogSim*. Comme illustré dans la Figure 19, ce composant donne accès aux trois interfaces suivantes.

1. *Formulation du problème :* cette interface offre aux utilisateurs la possibilité d'utiliser un ensemble de méthodes basées sur la programmation linéaire (ex. la génération de la fonction objectif et des contraintes) pour formuler le placement de données comme un problème de type GAP (voir la Section 4.4 du chapitre 4). Cette formulation consiste à affecter à chaque paire *(donnée, emplacement de stockage)* un coût de placement qui peut être exprimé en temps de latence de service, consommation énergétique, trafic réseau ou

tout autre métrique. La formulation du problème comprend un ensemble de contraintes de placement. Par exemple, on peut citer des contraintes pour limiter la quantité de données affectées à un ou à plusieurs emplacements de stockage. La formulation linéaire du problème de placement a la forme suivante :

$$\begin{cases} \text{Minimiser} & \sum_i \sum_j cout_{i,j} \times x_{i,j} \\ \text{Sous contraintes.} & \\ & contrainte\ 1 \\ & contrainte\ 2 \\ & \vdots \\ & x_{i,j} \in \{0,1\} \end{cases}$$

Avec :

- $cout_{i,j}$ représentant le coût de placement de la donnée i dans l'hébergeur j.
- $x_{i,j} = 1$ si la donnée i est placée dans l'hébergeur j, 0 sinon.
- $contrainte\ 1$ et $contrainte\ 2$ sont des contraintes de placement.

Toutefois, les utilisateurs peuvent ajouter d'autres contraintes dans cette formulation afin de personnaliser les fonctions de coût de placement.

2. *Résolution du problème :* une fois le problème de placement formulé, les utilisateurs peuvent utiliser l'interface *Résolution du problème* pour le résoudre. Cette interface donne accès au solveur de problèmes linéaires *CPLEX* MILP [6]. Avec cette interface, les utilisateurs peuvent appeler *CPLEX* avec différentes configurations comme la taille de la mémoire de travail utilisée pour résoudre le problème et le temps maximal pour trouver une solution. Néanmoins, les utilisateurs peuvent faire appel à d'autres solveurs de problèmes linéaires.

3. *Stratégies de placement prédéfinies :* cette interface donne la possibilité d'utiliser un ensemble de stratégies de placement mises en œuvre dans *iFogSim*, afin de comparer les différentes solutions, ou de les modifier pour en produire de nouvelles. Ces stratégies sont décrites ci-après :

   - **Cloud :** cette stratégie consiste à placer toutes les données dans des centres de données du *Cloud*. Les instances de services de l'IoT peuvent par la suite les récupérer et les traiter. Cette stratégie illustre le mode de stockage traditionnel des données de l'IoT dans le *Cloud*.
   - **Naïve :** avec cette stratégie, toutes les données sont stockées dans leur source si celle-ci possède un espace de stockage libre, sinon, elles sont

stockées dans l'hébergeur le plus proche de leur source en termes de latence.

- **iFogStor :** cette stratégie a été proposée dans le chapitre 4. Elle consiste à modéliser le placement de données comme un problème GAP puis de le résoudre en utilisant le solveur *CPLEX* MILP. Nous avons mis en œuvre cette stratégie en utilisant les deux interfaces *Formulation du problème* et *Résolution du problème*. Cette stratégie trouve la solution optimale pour stocker toutes les données en minimisant la latence globale du système. Au vu de la complexité du calcul de placement, cette stratégie peut causer un problème d'explosion combinatoire pour les grandes infrastructures (i.e. infrastructures avec des dizaines de milliers de nœuds).
- **iFogStorZ :** cette stratégie est une heuristique basée sur le concept de "diviser pour régner". Son objectif est de réduire le temps de calcul du placement de données. Cette stratégie a été déjà décrite dans la Section 4.5.2 du chapitre 4. Elle consiste à subdiviser l'infrastructure du système géographiquement en plusieurs zones constituant des sous-problèmes de moindre complexité. Ensuite, pour chaque zone, une instance d'*iFogStor* est lancée pour placer les données. Nous avons mis en œuvre cette stratégie en utilisant l'interface *Partitionnement géographique*, présentée dans la section suivante.
- **iFogStorG :** cette stratégie a fait l'objet du chapitre 5. Elle suit la même idée de partitionnement de l'infrastructure, mais celle-ci utilise des méthodes basées sur la théorie des graphes dans le processus de partitionnement, ceci afin de maintenir une qualité de placement élevée. Nous avons mis en œuvre cette stratégie en utilisant l'interface *Partitionnement par graphe* présentée dans la section suivante.

6.4 PARTITIONNEMENT DE L'INFRASTRUCTURE

Comme mentionné dans la Section 4.1 du chapitre 4, la gestion du placement de données dans le *Fog* est un problème *NP-difficile*. La résolution de ce type de problème peut prendre un temps important. D'autre part, le placement de données doit être faisable périodiquement et en cours d'exécution afin de s'adapter aux changements fréquents des infrastructures de type *Fog*. Pour ce faire, nous avons proposé deux heuristiques basées sur le concept de "diviser pour régner". Dans les deux heuristiques, l'infrastructure du système est subdivisée en plusieurs sous-parties constituant plusieurs sous-problèmes avec des complexités réduites. Ensuite, une stratégie de placement (ex. algorithme exact) est lancée pour placer les données dans chaque sous-partie d'une manière indépendante.

Afin de mettre en œuvre ces deux heuristiques, nous avons ajouté le composant *Partitionnement de l'infrastructure* à *iFogSim*. Ce composant offre deux méthodes pour subdiviser l'infrastructure en plusieurs sous-parties, voir la Figure 19.

6.4.1 *Partitionnement géographique*

Cette méthode subdivise l'infrastructure du système géographiquement en plusieurs zones (ex. quartiers, villes, régions, etc). Avec cette méthode, le temps de placement de données peut être significativement réduit. Cependant, cette méthode souffre de plusieurs inconvénients comme l'incapacité du passage à l'échelle et la dégradation de la qualité de placement. Pour plus de détails, voir le premier paragraphe du chapitre 5.

6.4.2 *Partitionnement par graphe*

Afin d'éviter les inconvénients causés par la subdivision géographique, nous avons mis en œuvre le composant *Partitionnement par graphe* qui utilise des méthodes basées sur la théorie des graphes pour partitionner l'infrastructure. Le processus de partitionnement est réalisé en 3 étapes : génération du graphe, pondération du graphe et partitionnement du graphe (voir les Sections 5.3.2, 5.4 et 5.5 du chapitre 5).

6.5 RÉPARTITION DES FLUX DE DONNÉES

Cette extension comprend une modélisation d'un scénario générique de "ville intelligente", afin de permettre aux utilisateurs de tester et d'évaluer leurs stratégies de placement de données. Comme cela a été montré dans la Section 5.7.4 du chapitre 5, la nature de distribution des flux de données a un impact sur la qualité de la stratégie de placement de données utilisée. Nous avons ajouté le composant *Répartition des flux de données* qui offre la possibilité de générer, pour notre scénario de simulation, les répartitions de données suivantes : zonée, distribuée et mixte. Ces répartitions sont décrites dans la section 5.7.1 du chapitre 5.

6.6 SIMULATION DE PLACEMENT DE DONNÉES

Dans cette partie, nous décrivons comment les données sont gérées dans *iFogSim*, sans et avec l'utilisation de notre extension. Le digramme de séquence de la Figure 20 montre la production, le traitement et le placement de données.

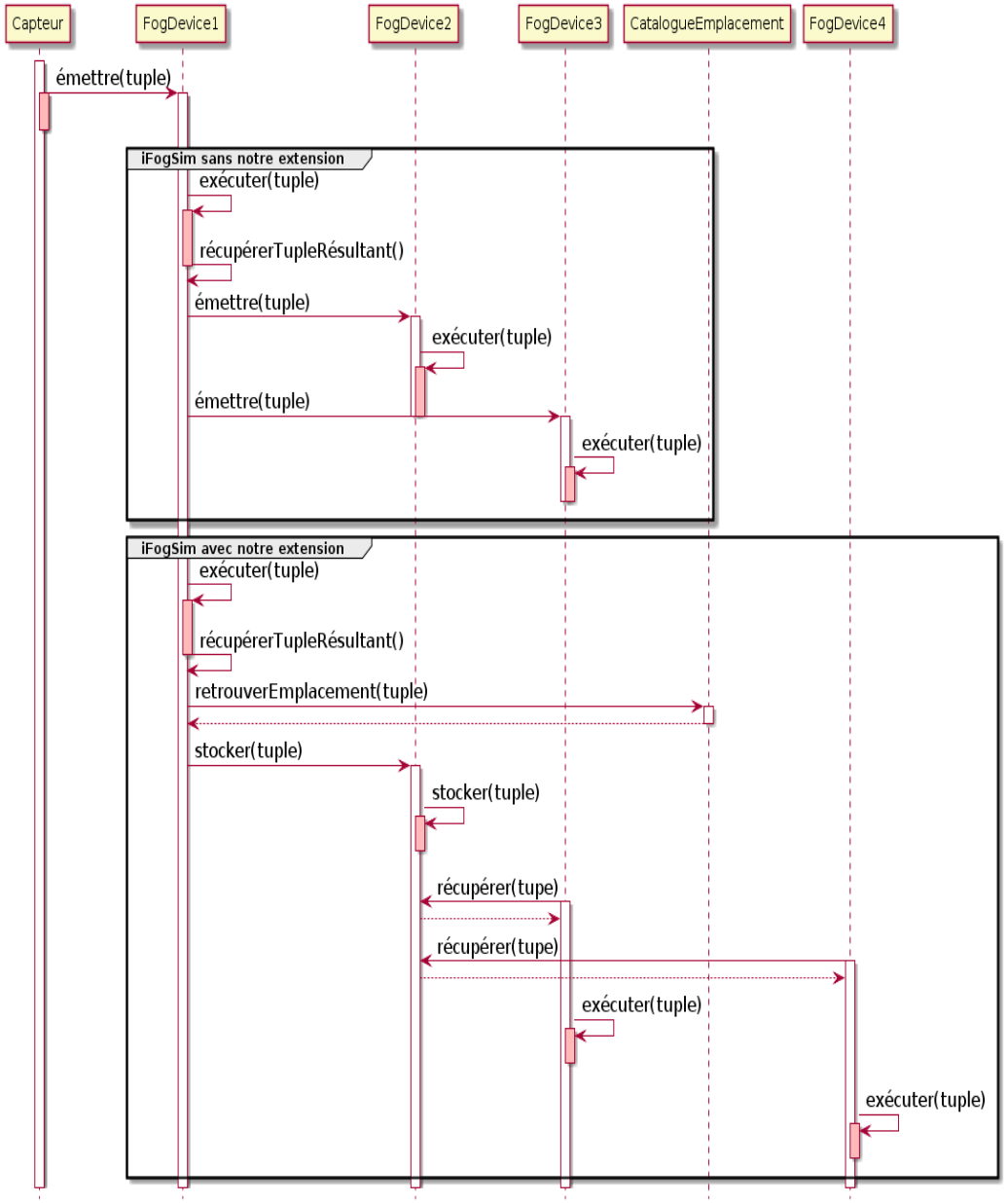


FIGURE 20 : Diagramme de séquence UML montrant la gestion de données dans *iFogSim*, sans et avec placement.

Tout d'abord, un capteur produit une donnée et la transmet à la passerelle associée (i.e. *FogDevice1*). Cette dernière manipule les données arrivées selon la version d'*iFogSim*.

- ***iFogSim* sans placement de données :** dans la version actuelle d'*iFogSim*, les données sont transférées directement aux consommateurs. Par exemple, comme illustré dans le diagramme de séquence, la passerelle traite les données arrivées du capteur et envoie les données résultantes aux consommateurs associés (i.e. *FogDevice2* et *FogDevice3*). Cette méthode génère plus de latence et consomme plus d'énergie car il faut envoyer les données autant

de fois qu'il y a de consommateurs même si ces derniers sont localisés au même endroit.

- ***iFogSim* avec placement de données :** en utilisant notre extension, les données produites sont stockées d'abord dans des nœuds de stockage suivant une stratégie de placement définie par l'utilisateur. Ensuite les nœuds consommateurs récupèrent les données requises depuis les nœuds de stockage et les traitent. Par exemple, comme illustré dans le diagramme de séquence, les données arrivant depuis le capteur sont traitées dans la passerelle associée. Ensuite, si le traitement produit des données de sortie, la passerelle interroge le catalogue des emplacements (*CatalogueEmplacement*) pour connaître les emplacements de stockage de ces données. Une fois les emplacements connus, la passerelle envoie les données aux nœuds de stockage spécifiés. Enfin, les consommateurs récupèrent les données depuis les nœuds de stockage et peuvent ainsi les traiter.

6.7 MISE EN ŒUVRE DE L'ALGORITHME DE FLOYD WARSHALL

Floyd Warshall [53] est un algorithme utilisé dans la théorie des graphes pour calculer tous les plus courts chemins dans un graphe pondéré. *iFogSim* utilise cet algorithme pour trouver le plus court chemin et la valeur de latence existante entre chaque paire de nœuds avant de commencer la simulation. Cela est fait afin de simuler le transfert de données entre les différents nœuds physiques de l'infrastructure. Quand le nombre de nœuds est grand (i.e. des milliers ou des millions de nœuds), le temps de calcul de cet algorithme peut être significatif. Cela est dû à sa complexité qui est en $O(n^3)$, avec n représentant le nombre total de nœuds du graphe [91]. Afin d'accélérer l'exécution de cet algorithme, et par conséquent, réduire le temps de simulation, nous avons développé une version parallélisée en utilisant la bibliothèque de calcul parallèle *OpenMP* [44] écrite en *C*. Avec cette version parallélisée, les utilisateurs peuvent spécifier le nombre de fils d'exécution (*Threads*) à utiliser afin d'avoir l'accélération souhaitée. Comme *iFogSim* est écrit en *Java*, nous avons utilisé l'interface de fonctions natives de *Java* (*Java Native Interface*) [38] pour faire appel à notre mise en œuvre de cet algorithme.

Le pseudo code de notre implantation de l'algorithme de *Floyd Warshall* est donné dans l'Algorithme 4.

n est le nombre de nœuds du graphe, et *dist* la matrice des distances existantes entre ces nœuds. Le coefficient *dist[i][j]* de cette matrice est initialisé par la latence du lien réseau direct entre le nœud *i* et le nœud *j* s'il existe, sinon, par ∞. À la fin de l'exécution de l'algorithme, *dist[i][j]* contient la valeur de la latence minimale

Algorithme 3 : L'algorithme de Floyd Warshall

Entrées : dist,flow
Output : dist, flow

```
1  function FloydWarshall()
2      pour 1 ⩽ k ⩽ n faire
3          #pragma omp parallel for private(i,j) // parallélisation de la
            deuxième et la troisième boucle de pour
4          pour 1 ⩽ i ⩽ n faire
5              si dist[i,j] ≠ ∞ et i ≠ k alors
6                  pour 1 ⩽ j ⩽ n faire
7                      si dist[i,k] + dist[k,j] < dist[i,j] alors
8                          dist[i,j] ← dist[i,k] + dist[k,j]
9                          flow[i,j] ← flow[k,j]
10     retourner dist,flow
```

entre le nœud i et le nœud j. Si aucun lien réseau n'est trouvé entre le nœud i et le nœud j, alors *dist[i][j]* =∞. flow est la matrice des nœuds prédécesseurs des nœuds finaux dans les chemins correspondant aux latences minimales (voir l'exemple donné ci-après). Elle spécifie pour chaque chemin optimal (en latence) possible, le nœud prédécesseur de son dernier nœud (i.e. le nœud prédécesseur du nœud d'arrivé). Par exemple, si P = A -> B -> C -> D est le plus court chemin existant entre le nœud A et le nœud D dans un graphe quelconque, alors *flow[A][D]* = C. À la fin de l'exécution de l'algorithme, si *flow[i][j]* =∞ cela signifie qu'aucun chemin réseau n'a été trouvé entre le nœud i et le nœud j. Dans l'autre cas, *flow[i][j]* =θ, ce qui signifie que le plus court chemin existant entre le nœud i et le nœud j a θ comme prédécesseur du nœud j. Le plus court chemin entre i et j est construit en remontant de j à i via les prédécesseurs stockés dans la matrice *flow*.

L'algorithme de *Floyd Warshall* est expliqué avec un exemple dans l'Annexe B.

6.8 ÉVALUATION ET RÉSULTATS

Dans cette section, nous décrivons d'abord la méthodologie utilisée pour évaluer notre extension du simulateur *iFogSim*, qui a pour objectif de fournir une plateforme expérimentale pour les travaux d'optimisation de placement de données dans le *Fog*. Ensuite, nous présentons les différentes configurations utilisées dans nos expérimentations. Nous finissons cette section par présenter et discuter des résultats d'expérimentations obtenus.

6.8.1 *Méthodologie d'évaluation*

Afin d'évaluer notre extension d'*iFogSim*, nous mesurons la surcharge ajoutée à *iFogSim* en termes de temps de simulation et d'utilisation mémoire par l'intégration de cette extension. De plus, nous valorisons l'accélération obtenue par l'utilisation de notre version parallélisée de l'algorithme de *Floyd Warshall*.

- **Temps de simulation :** nous avons mesuré le temps nécessaire pour faire une simulation du scénario défini dans la Section 4.6.1 du chapitre 4, sans et avec l'utilisation de notre extension de gestion de placement de données. Dans le premier cas, les données sont transmises directement depuis les producteurs aux consommateurs associés sans optimiser leur placement. À l'inverse, dans le deuxième cas, les données sont stockées d'abord dans les nœuds de *Fog* suivant la stratégie **naïve** de placement présentée dans la Section 4.6.3 du chapitre 4. Cette stratégie consiste à stocker les données dans leur source si celle-ci possède un espace de stockage libre, sinon dans le nœud de stockage le plus proche en terme de latence de leur producteur. Ensuite, les consommateurs peuvent récupérer puis traiter les données requises. Nous avons choisi cette stratégie car elle ne nécessite pas l'utilisation d'autres outils qui peuvent influencer la mesure des métriques choisies.

- **Empreinte mémoire :** comme pour le temps de simulation, nous avons mesuré la taille de la mémoire allouée pour simuler le scénario cité précédemment, sans et avec l'utilisation de notre extension de placement de données. Pour ce faire, nous avons utilisé le collecteur d'information sur l'activité du système *pidstat* disponible dans l'outil *Sysstat* [58].

- **Accélération du temps d'exécution de l'algorithme de *Floyd Warshall* :** nous avons mesuré le temps nécessaire pour calculer tous les chemins de latences minimales existant entre les différents nœuds de *Fog* en utilisant la version séquentielle (i.e. la version utilisée par défaut dans *iFogSim*) et notre version parallélisée. Pour cette dernière, nous avons fait varier le nombre de fils d'exécution (threads) utilisés de 1 à 32, et pour chaque cas, nous avons observé l'accélération obtenue.

6.8.2 *Configurations*

Afin d'évaluer notre extension, nous avons utilisé le même scénario de "ville intelligente" défini dans la Section 4.6.1 du chapitre 4. Nous avons simulé une infrastructure qui comprend 5 centres de données, 10 RPOP et 100 LPOP. Nous avons fait varier le nombre de passerelles simulées par 1000, 2000 et 3000, afin

d'étudier la surcharge ajoutée à *iFogSim* avec des complexités différentes du système. Nous avons fixé le nombre de capteurs et d'actionneurs par passerelle à 15 et à 5, respectivement. L'espace de stockage libre dans chaque nœud de *Fog*, et les latences existantes entre eux sont décrits respectivement dans la Table 3 et la Table 4 du chapitre 4. Afin d'évaluer notre version parallélisée de l'algorithme de *Floyd Warshall*, nous avons simulé une infrastructure qui comprend plus de 5000 nœuds de *Fog*.

Les simulations sont réalisées en utilisant un serveur à 48 cœurs de CPU *Xeon E5-2650* cadencés à 2.2 GHz et de 348 Go de RAM, en utilisant la distribution *Ubuntu* version 16.04 du système d'exploitation *GNU/Linux*.

6.8.3 *Résultats et discussion*

Dans cette section, nous discutons des résultats de simulation pour les métriques citées dans la section précédente.

- **Temps de simulation :** la Figure 21 montre le temps (en secondes) nécessaire pris par *iFogSim* pour simuler une infrastructure avec 1000, 2000 et 3000 passerelles, respectivement. Nous observons que le temps de simulation augmente d'une manière exponentielle avec la taille de l'infrastructure simulée. Par exemple, *iFogSim* a pris environ de 800 secondes pour simuler une infrastructure avec 1000 passerelles, alors qu'il a pris plus de 13400 secondes pour simuler une infrastructure avec 3000 passerelles.

  Concernant la surcharge en termes de temps de simulation ajoutée à *iFogSim* par notre extension, elle est d'environ 5%, 7% et 13% pour une infrastructure de 1000, 2000 et 3000 passerelles, respectivement (voir la Figure 21). Cette surcharge est due au traitement des événements supplémentaires ajoutés à *iFogSim* pour gérer le placement de données. Quelques exemples de ces événements sont celui permettant de retrouver le nœud de stockage d'une donnée et celui permettant de stocker une donnée, voir le diagramme de séquence dans la Figure 20.

- **Empreinte mémoire :** la Figure 22 montre la moyenne et le maximum de l'utilisation de la mémoire d'*iFogSim*, sans et avec notre extension. Nous pouvons observer que la gestion du placement de données a un impact faible sur l'empreinte mémoire. Cela est vrai pour les différentes configurations de l'infrastructure utilisées dans les simulations. Cette surcharge est due à l'allocation mémoire des événements supplémentaires ajoutés à *iFogSim* pour gérer le placement de données. D'autre part, la période d'utilisation maximale de la mémoire se produit pendant la phase d'initialisation de la simulation (i.e. envoi d'événements pour la création de l'infrastructure, etc.). En effet, notre extension intervient en dehors de cette phase. Par

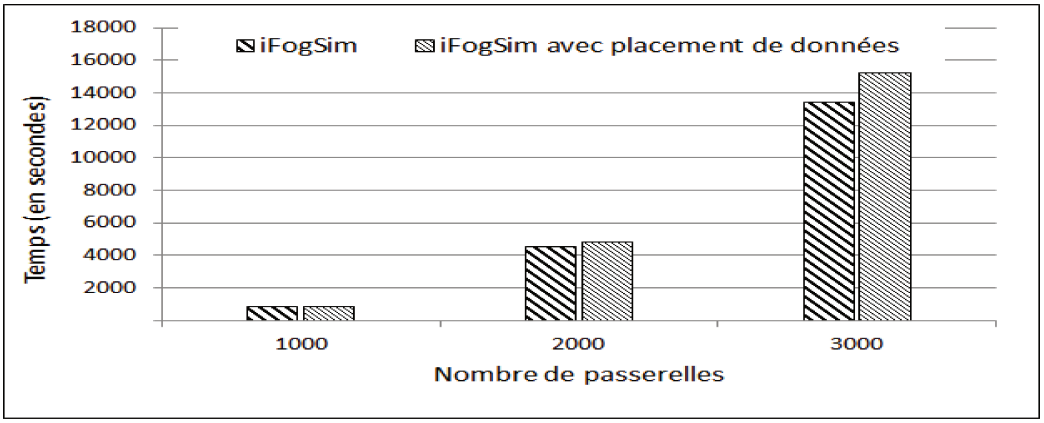


FIGURE 21 : Temps de simulation.

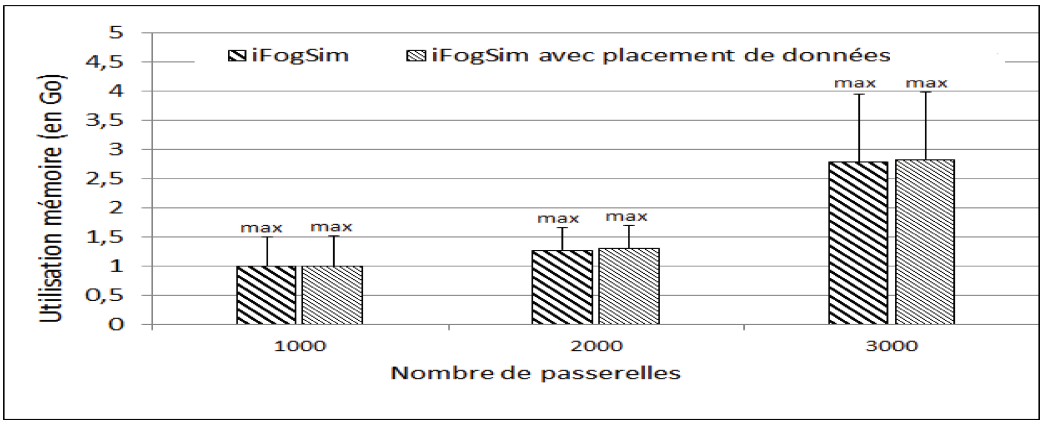


FIGURE 22 : Empreinte mémoire.

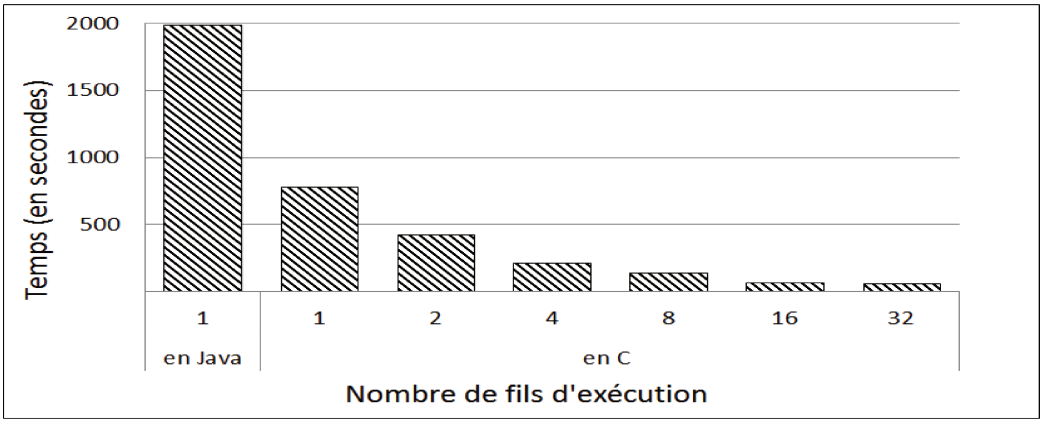


FIGURE 23 : Accélération de l'algorithme de *Floyd Warshall*.

ailleurs, nous remarquons qu'*iFogSim* prend jusqu'à 4 Go pour simuler une infrastructure contenant 3000 passerelles avec 15 capteurs et 5 actionneurs connectés à chaque passerelle, valeur compatible avec une machine dédiée à un usage général.

- **Accélération du temps d'exécution de l'algorithme de *Floyd Warshall* :** la Figure 23 montre le temps d'exécution (en secondes) de l'algorithme de *Floyd Warshall* pour calculer tous les plus courts chemins dans une infrastructure qui comprend 5000 nœuds de *Fog*. Le temps d'exécution est mesuré suivant deux mises en œuvre différentes de l'algorithme de *Floyd Warshall*. La première est la version séquentielle existant par défaut dans *iFogSim*, et la deuxième est la version parallélisée que nous proposons. Nous avons évalué la deuxième version en deux modes : (i) sans la parallélisation (avec la mise en œuvre en langage C) et (ii) avec parallélisation en faisant varier le nombre de fils d'exécution. La Table 9 illustre l'accélération obtenue par la version parallélisée pour le premier mode versus le deuxième mode d'évaluation. Comme montré dans la Figure 23, la version séquentielle de l'algorithme prend environ 2000 secondes pour trouver tous les plus courts chemins. Comme les infrastructures changent leur topologie fréquemment, ce temps d'exécution peut être considéré comme étant très important pour un déploiement en cours d'exécution. D'autre part, l'utilisation de notre version avec un seul fil d'exécution (i.e. le premier mode d'évaluation) réduit le temps d'exécution à 780 secondes, donnant une accélération de 2.5 (avec le jeux de données testé). Par ailleurs, comme illustré dans la Table 9, avec l'augmentation du nombre de fils d'exécution, le temps d'exécution a été accéléré de manière significative. En effet, en utilisant 32 fils d'exécution, l'algorithme prend environ 60 secondes pour trouver tous les plus courts chemins divisant ainsi le temps d'exécution par 12 en comparaison avec notre version sans parallélisation (i.e. le premier mode d'évaluation), et par 32 en comparaison avec la version séquentielle mise en œuvre par défaut dans *iFogSim*.

6.9 CONCLUSION ET PERSPECTIVES

Dans ce chapitre, nous avons présenté une extension du simulateur *iFogSim* permettant de modéliser et de simuler des scénarios avec des stratégies de placement de données. Cette extension permet aux utilisateurs de simuler leurs solutions pour gérer le placement de données dans les infrastructures de *Fog* au regard de la latence de service, de la consommation énergétique, du trafic réseau, et du coût opérationnel du système. L'extension consiste à rajouter 3 fonctionnalités à *iFogSim* : (i) le calcul des emplacements de données en utilisant des fonc-

TABLE 9 : Accélération obtenue par la version parallélisée de l'algorithme de *Floyd Warshall* versus l'utilisation d'un fils d'exécution.

Nombre de fils d'exécution	**Accélération**
2	1.84
4	3.65
8	5.42
16	11.83
32	12.99

tions basées sur la programmation linéaire, (ii) la subdivision de l'infrastructure en plusieurs sous-parties pour réduire le temps de placement, et (iii) un scénario générique de "ville intelligente" qui comprend plusieurs répartitions différentes des flux de données pour l'évaluation. L'extension inclut aussi la mise en œuvre d'une version parallèle de l'algorithme de *Floyd Warshall* pour réduire le temps de calcul des plus courts chemins existants entre les différents nœuds de l'infrastructure simulée. Comme évoqué dans la section d'évaluation, cette extension a une surcharge minime sur le temps de simulation et sur l'empreinte mémoire d'*iFogSim*.

Une direction intéressante pour les futurs travaux est d'élargir cette extension pour gérer la réplication et la cohérence de données dans les infrastructures de type *Fog*. En effet, le stockage de plusieurs réplicats de donnée offre la possibilité : (i) d'augmenter la résilience contre la perte de données et (ii) de réduire la latence du système par la parallélisation des requêtes concurrentes sur les différents réplications.

Troisième partie

CONCLUSION

Chapitre 7

CONCLUSION

Ce chapitre conclut le travail de cette thèse. Nous y faisons d'abord un rappel des différentes contributions faites pour optimiser le placement de données de l'IoT dans les infrastructures de type *Fog*. Nous finissons ce chapitre par décrire quelques perspectives pour les travaux futurs.

Sommaire

7.1 CONTRIBUTIONS

Les infrastructures de *Fog* étant denses, hétérogènes et géo-distribuées, le déploiement des services d'IoT dans ces infrastructures exige de prendre en considération le placement de données afin de réduire la latence dans le système. À cette fin, dans ce travail de thèse, nous avons fait trois contributions décrites comme suit :

7.1.1 *Formulation du problème de placement de données dans le Fog*

Notre première contribution consiste en la formulation du problème de placement de données de l'IoT dans le *Fog* comme un problème d'affectation généralisé. Dans cette formulation, nous avons proposé un programme linéaire dont la fonction objectif minimise la latence globale du système, et les contraintes assurent que (i) chaque donnée est stockée dans un seul emplacement (il n'y a pas de réplication ni de segmentation de données dans notre système) et (ii) les capacités de stockage des nœuds de *Fog* ne doivent pas être dépassées.

7.1.2 *Stratégies de placement de données dans le Fog*

Pour optimiser la latence globale du système, nous avons proposé trois stratégies pour placer les données dont une approche exacte et deux heuristiques. Ces stratégies de placement sont décrites comme suit :

7.1.2.1 *Approche exacte, iFogStor*

Cette stratégie considère la totalité de l'infrastructure du système lors du calcul du placement. *iFogStor* trouve les emplacements optimaux pour stocker toutes les données en minimisant la latence globale du système. Cette stratégie explore directement le programme linéaire de la formulation du problème de placement que nous avons défini. En effet, elle fait appel au solveur de problèmes linéaires *CPELX* MILP pour calculer les emplacements de données. Les expérimentations ont montré qu'en utilisant *iFogStor*, le temps de latence du système est réduit par plus de 86% en comparaison avec une approche de placement dans le *Cloud*, et par plus de 60% en comparaison avec une approche naïve de placement dans le Fog. En raison des temps de résolution engendrés, nous avons proposé les deux heuristiques décrites ci-dessous pour réduire le temps de calcul du placement.

7.1.2.2 *Heuristique 1, iFogStorZ*

Comme mentionné auparavant, afin de s'adapter aux changements des infrastructures de *Fog*, le placement de données doit être fait périodiquement et en

ligne. À cette fin, nous avons proposé *iFogStorZ*, une heuristique basée sur le concept "diviser pour régner". *iFogStorZ* subdivise l'infrastructure du système en plusieurs zones géographiques. Pour chaque zone, un problème de placement de taille moindre est modélisé et résolu de manière indépendante des autres problèmes, et suivant la stratégie *iFogStor*. Au vu de la complexité du problème de placement d'origine, la subdivision de l'infrastructure permet une réduction considérable du temps de placement. De plus, le partitionnement de l'infrastructure permet de calculer les emplacements de données pour une ou plusieurs parties (parties dont les changements sont fréquents) plutôt que de les calculer pour toute l'infrastructure, réduisant ainsi le temps de placement. Les expérimentations ont montré qu'en utilisant *iFogStorZ*, le temps de placement de données est divisé par 117 par rapport à *iFogStor* en gardant l'optimalité à près de 96%. Ces résultats ont été obtenus en utilisant 10 zones géographiques et un jeu de données de type *zoné*, c'est-à-dire que les données sont utilisées par des services consommateurs localisés dans les mêmes zones géographiques que les producteurs.

7.1.2.3 *Heuristique 2, iFogStorG*

La subdivision géographique de l'infrastructure peut générer des flux de données inter-partition (ou inter-zone géographique) lorsque les consommateurs de données sont situés dans des partitions autres que celles des producteurs. Dans ce cas, le processus de placement (lancé dans une zone géographique) est inconscient de la localisation des consommateurs. Ainsi, les données ne peuvent pas être placées d'une manière optimale. Cela dégrade la qualité de la solution de placement trouvée. Cette dégradation s'aggrave avec l'augmentation du nombre de zones du fait de la génération des flux de données inter-partition supplémentaires. De plus, cette méthode de subdivision (géographique) peut générer des zones déséquilibrées en terme de temps de calcul. Ceci augmente le temps global du placement de données du fait qu'il dépende du temps de placement de la zone la plus grande. Par ailleurs, cette méthode ne passe pas l'échelle car le nombre total de zones possible est égal au nombre de zones géographiques. Cela limite la réduction du temps de placement.

Afin d'éviter les problèmes mentionnés ci-dessus, nous avons proposé *iFogStorG*, une heuristique qui suit la même logique d'*iFogStorZ*, mais qui utilise des méthodes basées sur la théorie des graphes pour partitionner l'infrastructure du système. *iFogStorG* essaye de garder autant que possible les producteurs et leurs consommateurs associés dans les mêmes zones lors du partitionnement de l'infrastructure. Cela permet de minimiser la perte d'information sur l'espace de recherche de la solution, et par conséquent, de réaliser un placement de données efficace. De plus, *iFogStorG* forme des zones équilibrées en termes de nombre de nœuds de *Fog* et de quantité de données. Cela permet de générer des

sous-problèmes de placement de complexité équivalente, et par conséquent, de réduire le temps total de placement. Par ailleurs, cette heuristique passe l'échelle en générant un nombre de partitions entre 1 et le nombre de nœuds de *Fog* de l'infrastructure du système. Les expérimentations ont montré qu'en utilisant *iFogStorG*, le temps de placement est divisé par 470 en comparaison avec *iFogStor* et en gardant l'optimalité de la solution près de 92% quel que soit le type de jeu de données utilisé (zoné ou distribué).

7.1.3 *Extension d'iFogSim avec la gestion de placement de données*

Afin de satisfaire le besoin en outils pour évaluer des stratégies de placement de données dans le *Fog*, nous avons étendu *iFogSim*. Cette extension consiste à ajouter un support de gestion de placement de données au simulateur d'environnements de *Fog* et d'IoT *iFogSim*. Nous avons ajouté à *iFogSim* trois fonctionnalités : (i) le calcul des emplacements de données en utilisant des fonctions basées sur la programmation linéaire, (ii) la subdivision de l'infrastructure en plusieurs sous-parties pour réduire le temps de placement, et (iii) un scénario générique de "ville intelligente" qui comprend plusieurs répartitions différentes des flux de données pour l'évaluation.

7.2 PERSPECTIVES

Les stratégies de placement de données proposées dans ce travail de thèse ont conduit à des résultats améliorant les latences dans les infrastructures de *Fog*, néanmoins, nous pensons qu'il est possible de faire mieux. Nous avons étudié le placement de données pour minimiser la latence globale du système sans avoir investigué le gain en latence apporté par la réplication de données. En effet, avec l'utilisation de cette dernière, la latence du système peut être réduite. Par exemple, chaque requête d'accès aux données peut-être desservie par le réplicat de données le plus proche du consommateur en termes de temps de latence. En revanche, l'ajout des réplicats de données exige de préserver de la cohérence de données dans le système. La gestion de la cohérence de données peut engendrer une latence de synchronisation des réplicats dans le système qui varie selon le niveau de cohérence demandé, le nombre de réplicats et leur emplacement.

7.2.1 *Formulation du problème de placement des réplicats de données dans le Fog*

La première contribution à faire est de formuler le problème du placement des réplicats de données dans le *Fog* pour minimiser la latence globale du système.

Ce problème consiste à trouver, pour chaque donnée existante dans le système, le nombre de réplicats nécessaires à produire ainsi que leurs emplacements de stockage afin de minimiser la latence globale du système, tout en respectant les exigences en cohérence de données. De plus, afin d'éviter des problèmes dus à la perte de données (ex. l'indisponibilité du service), des contraintes sur la disponibilité des nœuds de stockage et des liens réseaux peuvent être considérées dans cette formulation en fonction de la criticité des données et de la QoS. Néanmoins, dans une telle infrastructure comprenant un grand nombre de nœuds de *Fog*, de services d'IoT et d'une quantité massive de données, le trafic réseau, le coût de stockage et la consommation énergétique devraient être élevés. Pour cela, d'autres objectifs comme la minimisation du trafic réseau, le coût de stockage et la consommation énergétique peuvent être considérés (accompagnés de la latence) dans la formulation de problème.

7.2.2 *Stratégies du placement de réplicats de données dans le Fog*

Comme nous avons vu dans le chapitre 5, les stratégies de placement de données doivent être conçues pour un déploiement en ligne. Cela permet de s'adapter aux changements du système. Ces changements ont une influence sur la latence du système et peuvent rendre la solution antérieure du placement obsolète. Cela exige de calculer à nouveau les emplacements de données. Par ailleurs, l'ajout de la réplication de données et la gestion de la cohérence complexifie davantage le système. Les approches exactes risquent d'être inopérantes. Des solutions à base d'heuristiques sont donc à développer pour gérer le placement des réplicats et la cohérence de données dans le système.

7.2.3 *Extension d'iFogSim*

Afin de pouvoir évaluer des stratégies de réplication et de gestion de cohérence de données dans les infrastructures de *Fog*, il est nécessaire de fournir une plateforme expérimentale pour que les utilisateurs déploient, testent et évaluent leurs stratégies. Une piste possible est d'étendre le simulateur *iFogSim*.

Une autre direction intéressante pour les travaux futurs pour cette contribution consiste à permettre à cette extension de faire des simulations de manière parallèle que ce soit sur une seule machine ou dans un cluster, réduisant ainsi le temps de simulation. En effet, ce temps peut être très important dans le cas d'infrastructures *Fog* de grande taille (par exemple, *iFogSim* prend 3,72 heures pour simuler une infrastructure avec 3000 nœuds de *Fog*), comme indiqué dans la partie évaluation du chapitre 6.

7.2.4 *Placement de données et d'instances de services*

Une autre direction intéressante pour les contributions futures, est d'investiguer comment optimiser le placement d'instances de services ainsi que les données d'une façon conjointe afin de réduire la latence du système. En effet, comme nous avons vu dans le Chapitre 3, plusieurs travaux de l'état de l'art ont adressé le problème de placement des instances de service. Ces travaux ont conduit à une réduction de la latence du service. Par ailleurs, nous avons vu dans cette thèse que la latence de service peut aussi être réduite en gérant le placement de données. De notre point de vu, la latence du système pourrait encore être réduite en optimisant à la fois le placement des instances de services et des données, car le temps de latence dépend des emplacements des deux.

7.3 FIN

Nous clôturons ce travail de thèse par la citation suivante [5] :

« Les engagements se révèlent aux gens déterminés,
Les actes nobles se révèlent aux gens distingués,
Les petites difficultés s'agrandissent dans les yeux des faibles,
tandis que
Les calamites s'affaiblissent dans les yeux des grands !»

Al-mutanabbi.

عَلَى قَدْرِ أَهلِ العَزمِ تَأْتِي العَزَائِمُ

وَتَأْتِي عَلَى قَدرِ الكِرَامِ المَكَارِمُ

وَتَعْظُمُ فِي عَينِ الصَغِيرِ صِغَارُهَا

وَتَصغُرُ فِي عَينِ العَظِيمِ العَظَائِمُ

المتنبى

Quatrième partie

ANNEXE

Annexe *A*

NOTATIONS

Table 10 : Dictionnaire de notations 1

Notation	Description
S	L'infrastructure du système
fn	Nœud de Fog
dp_i	DataProd, une entité qui produit des données
dh_j	DataHost, une entité qui héberge des données
dc_k	DataCons, une entité qui consomme des données
FN	L'ensemble des nœuds du Fog
DH	L'ensemble des hébergeurs de données
DP	L'ensemble des producteurs de données
DC	L'ensemble des consommateurs de données
d_i	La donnée produite par dp_i
s_{d_i}	La taille de d_i (en octets)
D	L'ensemble des données produites par DP
A_D	La matrice d'affectation de D dans DH
$a_{i,j}$	= 1 si d_i est affectée à dh_j
f_{dh_j}	La capacité de stockage libre de dh_j
G	Graphe
v	Sommet du graphe
e	Arête du graphe
V	L'ensemble des sommets
E	L'ensemble des arêtes
P	L'ensemble des partitions de graphe
C	L'ensemble des arêtes coupées

TABLE 11 : Dictionnaire de notations 2

Notation	Description
$Overall\_Latency$	La latence globale du système générée par le stockage de D dans DH et sa récupération par DC
b	La granularité des échanges de données
TSD	Matrice contenant les temps de stockage de D depuis DP jusqu'à DH
$ts_{i,j}$	Le temps de transfert (stockage) b depuis dp_i jusqu'à dh_j
$ts_{d_{i,j}}$	Le temps de transfert (stockage) d_i depuis dp_i jusqu'à dh_j
TRD	Matrice contenant les temps de récupération de D depuis DH jusqu'à DC
$D_{k,j}$	Le sous-ensemble de données requises par dc_k depuis dh_j
$s_{D_{k,j}}$	La taille de $D_{k,j}$ (en octets)
$tr_{k,j}$	Le temps de transfert de b depuis dh_j jusqu'à dc_k
$tr_{D_{k,j}}$	Le temps de transfert de $D_{k,j}$ depuis dh_j jusqu'à dc_k
f	Flux de données, dépendance de données entre les producteurs et les consommateurs
DFM	Matrice contenant tous les flux de données
ADM	Matrice d'adjacence contenant les latences des liens physiques existants entre les nœuds
DPM	Matrice des emplacements des producteurs dans l'infrastructure
DCM	Matrice des emplacements des consommateurs dans l'infrastructure

Annexe B

ALGORITHMES

B.1 L'ALGORITHME DE FLOYD WARSHALL

L'algorithme de *Floyd-Warshall* permet de trouver les distances des plus courts chemins entre toutes les paires de sommets dans un graphe orienté et pondéré, en temps cubique en le nombre de sommets [91]. La distance entre deux sommets dans un graphe est calculée par la somme des poids des arcs constituant le chemin entre ces deux sommets.

Formulé autrement, soit $G(X, S)$ un graphe orienté et pondéré avec X étant l'ensemble des arêtes et S l'ensemble des sommets. Pour tout couple de sommets $(i, j) \in S^2$, l'algorithme détermine s'il existe un chemin de i à j, calcule sa longueur et construit une table de routage. Cet algorithme prend en entrée la matrice d'adjacence donnant le poids d'un arc lorsqu'il existe et la valeur ∞ sinon, c'est-à-dire une matrice $D_{i,j}$ de taille $N \times N$, où $N = |S|$, telle que $D_{i,j} < \infty$ si et seulement si $(i, j) \in S$. $D_{i,j}$ est alors le poids de l'arête (i, j). L'algorithme produit en sortie deux matrices W et F, où $W_{i,j}$ est la distance du plus court chemin entre i et j (s'il existe, $D_{i,j} = \infty$ sinon) et $F_{i,j}$ est le sommet prédécesseur de j dans le plus court chemin. Ce dernier est construit par un appel récursif de la matrice F (voir l'exemple ci-après).

L'algorithme *Floyd-Warshall* est un exemple de programmation dynamique. Il est constitué de N itérations principales ; pour chaque itération k, l'algorithme calcule les plus courts chemins entre toute paire de sommets avec des sommets intermédiaires appartenant uniquement à l'ensemble $1, 2, 3, \ldots, k$. À l'initialisation, l'algorithme calcule le plus court chemin entre toute paire de sommets n'ayant pas de sommets intermédiaires, il suffit donc de prendre la distance des arcs qui existent et mettre un poids infini si l'arc n'existe pas. Par la suite, si l'on note $M^k_{ij,}$ la valeur du plus court chemin de i à j dont les seuls sommets intermédiaires sont dans l'ensemble $1, 2, 3, \ldots, k$, alors on a l'égalité suivante :

$$M^k_{i,j} = \min(M^{k-1}_{i,j}, M^{k-1}_{i,k} + M^{k-1}_{k,j}) \tag{19}$$

Le pseudo code de l'algorithme de *Floyd Warshall* est donné dans l'Algorithme 4.

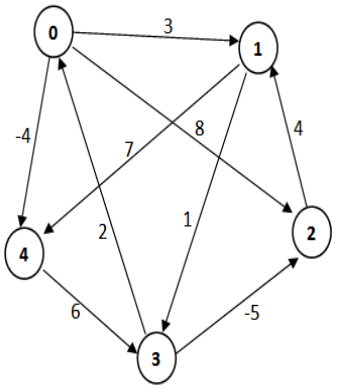
Algorithme 4 : L'algorithme de Floyd Warshall

Entrées : D

Output : W, F

```
1  Algorithme Floyd-Warshall ()
                                                    // Initialisation
2      pour 1 ⩽ i ⩽ N faire
3          pour 1 ⩽ j ⩽ N faire
4              W[i, j] ← D[i, j]
5              si D[i, j] ≠ ∞ alors
6                  F[i, j] ← i
7              sinon
8                  F[i, j] ← ∞
                                          // Calcul de plus courts chemins
9      pour 1 ⩽ k ⩽ N faire
10         pour 1 ⩽ i ⩽ N faire
11             si W[i, k] ≠ ∞ et i ≠ k alors
12                 pour 1 ⩽ j ⩽ N faire
13                     si W[i, k] + W[k, j] < W[i, j] alors
14                         W[i, j] ← W[i, k] + W[k, j]
15                         F[i, j] ← F[k, j]
16     retourner D,R
```

B.2 EXEMPLE



$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

FIGURE 24 : Exemple de graphe d'entrée de l'algorithme de *Floyd-Warshall*.

La Figure 24 illustre un exemple de graphe et sa matrice d'adjacence (D). Les différents étapes du calcul des plus courts chemins de ce graphe par l'algorithme de *Floyd Warshall* sont données ci-après.

$$W_0 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad F_0 = \begin{bmatrix} 0 & 0 & 0 & \infty & 0 \\ \infty & 1 & \infty & 1 & 1 \\ \infty & 2 & 2 & \infty & \infty \\ 3 & \infty & 3 & 3 & \infty \\ \infty & \infty & \infty & 4 & 4 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad F_1 = \begin{bmatrix} 0 & 0 & 0 & \infty & 0 \\ \infty & 1 & \infty & 1 & 1 \\ \infty & 2 & 2 & \infty & \infty \\ 3 & 0 & 3 & 3 & 0 \\ \infty & \infty & \infty & 4 & 4 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad F_2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ \infty & 1 & \infty & 1 & 1 \\ \infty & 2 & 2 & 1 & 1 \\ 3 & 0 & 3 & 3 & 0 \\ \infty & \infty & \infty & 4 & 4 \end{bmatrix}$$

$$W_3 = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad F_3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ \infty & 1 & \infty & 1 & 1 \\ \infty & 2 & 2 & 1 & 1 \\ 3 & 2 & 3 & 3 & 0 \\ \infty & \infty & \infty & 4 & 4 \end{bmatrix}$$

$$W_4 = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \quad F_4 = \begin{bmatrix} 0 & 0 & 3 & 1 & 0 \\ 3 & 1 & 3 & 1 & 0 \\ 3 & 2 & 2 & 1 & 0 \\ 3 & 2 & 3 & 3 & 0 \\ 3 & 2 & 3 & 4 & 4 \end{bmatrix}$$

$$W_5 = \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \quad F_4 = \begin{bmatrix} 0 & 2 & 3 & 4 & 0 \\ 3 & 1 & 3 & 1 & 0 \\ 3 & 2 & 2 & 1 & 0 \\ 3 & 2 & 3 & 3 & 0 \\ 3 & 2 & 3 & 4 & 4 \end{bmatrix}$$

RÉFÉRENCES

[1] Glpk , le solveur des programmes linéaires de gnu. https ://www.gnu.org/software/glpk/. Visitée : 2018-09-11.

[2] Gurobi , le solveur des programmes linéaires de microsoft. http ://www.gurobi.com/. Visitée : 2018-09-11.

[3] Internet protocol ipv6. http ://www.grm94.polymtl.ca/ lepage/IPv6/avril97. Visitée : 2018-08-24.

[4] Or-tools, le solveur des programmes linéaires de google. https ://developers.google.com/optimization/. Visitée : 2018-09-11.

[5] Almutanabbi. https ://fr.wikipedia.org/wiki/Al-Mutanabbi. Visitée : 2018-10-30.

[6] Cplex toolkit. http ://www.ibm.com/support/knowledgecenter/SSSA5P. Visitée : 2016-11-21.

[7] Google glass. https ://www.futura-sciences.com/tech/definitions/smartphone-google-glass-15803/. Visitée : 2018-08-23.

[8] Lab-sticc. https ://www.labsticc.fr/en/index/. Visitée : 2018-12-10.

[9] Metis, a graph partition tool. http ://glaros.dtc.umn.edu/gkhome/metis/metis/overview. Visitée : 2017-07-19.

[10] Number of iot developers in 2020. https ://readwrite.com/2014/06/27/internet-of-things-developers-jobs-opportunity/. Visitée : 2018-08-28.

[11] Smart eye. https ://iatranshumanisme.com/2018/01/23/smart-farms-vers-nouvelle-forme-dagriculture/. Visitée : 2018-08-23.

[12] Orange. https ://fr.statista.com/themes/3851/orange/, . Visitée : 2018-12-13.

[13] Orange et l'iot. https ://www.orange.com/fr/Human-Inside/Dossier-thematique/Les-promesses-de-l-IoT, . Visitée : 2018-12-13.

[14] Regional and local pops. http ://www.gatoux.com/SECTION4/p3.php. Visitée : 2016-12-21.

[15] Ruggedpod. https ://www.indiegogo.com/projects/ruggedpod-outdoor-cloud-computing. Visitée : 2016-12-21.

[16] Drone de santé. https ://club-digital-sante.info/2016/11/drones-sante-secours. Visitée : 2018-08-22.

[17] Skydrone. https ://www.skydrone.fr/transport-objet-par-drone. Visitée : 2018-08-22.

[18] Waze. https ://www.waze.com/fr, . Visitée : 2018-08-22.

[19] Waze stats. https ://searchengineland.com/waze-launches-local-ads-primarily-aimed-at-smbs-and-franchises-295285, . Visitée : 2018-08-22.

[20] J.Y. Girard A. Turing. *La machine de Turing*. Éditions du Seuil, 1995. ISBN 2-02-036928-1.

[21] M. Aazam and E. Huh. E-hamc : Leveraging fog computing for emergency alert service. In *Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on*, pages 518–523. IEEE, 2015.

[22] M. Abu-Elkheir, M. Hayajneh, and N. Ali. Data management for the internet of things : Design primitives and solution. *Sensors*, 13(11) :15582–15612, 2013.

[23] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne. Fit iot-lab : A large scale open experimental iot testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464, 2015. doi : 10.1109/WF-IoT.2015.7389098.

[24] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things : A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4) :2347–2376, Fourthquarter 2015. ISSN 1553-877X.

[25] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou. On uncoordinated service placement in edge-clouds. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 41–48, Dec 2017. doi : 10.1109/CloudCom.2017.46.

[26] H. Atlam, R. Walters, and G. Wills. Fog computing and the internet of things : A review. *Big Data and Cognitive Computing*, 2(2), 2018. ISSN 2504-2289. doi : 10.3390/bdcc2020010. URL http://www.mdpi.com/2504-2289/2/2/10.

[27] L. Atzori, A. Iera, and G. Morabito. The internet of things : A survey. *Comput. Netw.*, 54(15) :2787–2805, October 2010. ISSN 1389-1286. doi : 10.1016/j.comnet.2010.05.010. URL http://dx.doi.org/10.1016/j.comnet.2010.05.010.

[28] S.R. Balachandar and K. Kannan. A new heuristic approach for the large-scale generalized assignment problem. *Int J Math Comput Phys Elect Comput Eng*, 3 :969–974, 2009.

[29] S.R. Balachandar and K. Kannan. A new heuristic approach for the large-scale generalized assignment problem. *International Journal of Mathematical and Statistical Sciences*, 1(4), 2009.

[30] H. Bangui, S. Rakrak, S. Raghay, and B. Buhnova. Moving to the edge-cloud-of-things : Recent advances and future research directions. *Electronics*, 7(11) :309, 2018.

[31] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1519-7. doi : 10.1145/2342509.2342513. URL http://doi.acm.org/10.1145/2342509.2342513.

[32] A. Brogi and S. Forti. Qos-aware deployment of iot applications through the fog. *IEEE Internet of Things Journal*, 4(5) :1185–1192, Oct. 2017 2017. ISSN 2327-4662. doi : 10.1109/JIOT.2017.2701408.

[33] C. Byers and P. Wetterwald. Fog computing distributing data and intelligence for resiliency and scale necessary for iot : The internet of things (ubiquity symposium). *Ubiquity*, 2015 :4 :1–4 :12, November 2015. ISSN 1530-2180. doi : 10.1145/2822875. URL http://doi.acm.org/10.1145/2822875.

[34] C. Byers and P. Wetterwald. Fog computing distributing data and intelligence for resiliency and scale necessary for iot : The internet of things (ubiquity symposium). *Ubiquity*, 2015(November) :4, 2015.

[35] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A. De Rose, and R. Buyya. Cloudsim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software : Practice and experience*, 41(1) :23–50, 2011.

[36] D.G. Cattrysse and L.N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60(3) :260 – 272, 1992. ISSN 0377-2217. doi : https://doi.org/10.1016/0377-2217(92)90077-M. URL http://www.sciencedirect.com/science/article/pii/037722179290077M.

[37] C. Cecchinel, M. Jimenez, S. Mosser, and M. Riveill. An architecture to support the collection of big data in the internet of things. In *2014 IEEE World Congress on Services*, pages 442–449, June 2014. doi : 10.1109/SERVICES.2014.83.

[38] M. Chen, S. Goldenberg, S. Srinivas, V. Ushakov, Y. Wang, Q. Zhang, E. Lin, and Y. Zach. Java jni bridge : A framework for mixed native isa execution. In *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '06, pages 65–75, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2499-0. doi : 10.1109/CGO.2006.22.

[39] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs. Building a big data platform for smart cities : Experience and lessons from santander. In *2015 IEEE International Congress on Big Data*, pages 592–599, June 2015. doi : 10.1109/BigDataCongress.2015.91.

[40] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs. Building a big data platform for smart cities : Experience and lessons from santander. In *2015 IEEE International Congress on Big Data*, pages 592–599, June 2015. doi : 10.1109/BigDataCongress.2015.91.

[41] P.C. Chu and J.E. Beasley. A genetic algorithm for the generalised assignment problem. *Computers and Operations Research*, 24(1) :17 – 23, 1997. ISSN 0305-0548. doi : https://doi.org/10.1016/S0305-0548(96)00032-9. URL http://www.sciencedirect.com/science/article/pii/S0305054896000329.

[42] Fog Computing. the internet of things : Extend the cloud to where the things are. *Cisco White Paper*, 2015.

[43] A. Coutinho, F. Greve, C. Prazeres, and J. Cardoso. Fogbed : A rapid-prototyping emulation environment for fog computing. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

[44] L. Dagum and R. Menon. Openmp : an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1) : 46–55, 1998. ISSN 1070-9924. doi : 10.1109/99.660313.

[45] N. Daneshfar, N. Pappas, V. Polishchuk, and V. Angelakis. Service allocation in a mobile fog infrastructure under availability and qos constraints. *arXiv preprint arXiv :1706.04084*, 2017.

[46] N. Daneshfar, N. Pappas, V. Polishchuk, and V. Angelakis. Service Allocation in a Mobile Fog Infrastructure under Availability and QoS Constraints. *ArXiv e-prints*, June 2017.

[47] A.V. Dastjerdi, H. Gupta, R.N. Calheiros, S.K. Ghosh, and R. Buyya. Fog computing : Principles, architectures, and applications. *CoRR*, abs/1601.02752, 2016. URL http://arxiv.org/abs/1601.02752.

[48] J.L. Deneubourg, S. Aron, S. Goss, and J.M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior*, 3(2) : 159–168, 1990.

[49] A. Di Nardo, M. Di Natale, G.F. Santonastaso, V. Tzatchkov, and V.H. Yamanaka. Divide and conquer partitioning techniques for smart water networks. *Procedia Engineering*, 89 :1176–1183, 2014.

[50] M. Douiri, S. Elbernoussi, and H. Lakhbab. Cours des méthodes de résolution exactes heuristiques et métaheuristiques. *Université Mohamed V, Faculté des sciences de Rabat*, pages 5–87, 2009.

[51] C. Dsouza, G. Ahn, and M. Taguinod. Policy-driven security management for fog computing : Preliminary framework and a case study. In *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*, pages 16–23, Aug 2014. doi : 10.1109/IRI.2014.7051866.

[52] M. Etemad, M. Aazam, and M. St-Hilaire. Using devs for modeling and simulating a fog computing environment. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 849–854, 2017. doi : 10.1109/ICCNC.2017.7876242.

[53] R.W. Floyd. Algorithm 97 : shortest path. *Communications of the ACM*, 5(6) : 345, 1962.

[54] P. Fouilhoux. Optimisation combinatoire : Programmation linéaire et algorithmes. *Université Pierre et Marie Curie*, 2015.

[55] F. Ganz, R. Li, P. Barnaghi, and H. Harai. A resource mobility scheme for service-continuity in the internet of things. In *2012 IEEE International Conference on Green Computing and Communications*, pages 261–264, Nov 2012. doi : 10.1109/GreenCom.2012.48.

[56] C. García, D. Meana-Llorián, J. Lovelle, et al. A review about smart objects, sensors, and actuators. *International Journal of Interactive Multimedia & Artificial Intelligence*, 4(3), 2017.

[57] V. Gazis, A. Leonardi, K. Mathioudakis, K. Sasloglou, P. Kikiras, and R. Sudhaakar. Components of fog computing in an industrial internet of things context. In *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops)*, pages 1–6, June 2015. doi : 10.1109/SECONW.2015.7328144.

[58] S. Godard. Sysstat utilities home page. *sebastien.godard.pagesperso-orange.fr*, 2010. Visitée : 2017-11-10.

[59] growthEnabler. Market pulse report, internet of things (iot). 2017.

[60] B. Gu, Y. Chen, H. Liao, Z. Zhou, and D. Zhang. A distributed and context-aware task assignment mechanism for collaborative mobile edge computing. *Sensors*, 18(8) :2423, 2018.

[61] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot) : A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7) :1645–1660, 2013.

[62] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, and R. Buyya. iFogSim : A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments. *ArXiv e-prints*, June 2016.

[63] P. Hu, S. Dhelim, H. Ning, and T. Qiu. Survey on fog computing : architecture, key technologies, applications and open issues. *Journal of Network and Computer Applications*, 98 :27 – 42, 2017. ISSN 1084-8045. doi : https://doi.org/10.1016/j.jnca.2017.09.002. URL http://www.sciencedirect.com/science/article/pii/S1084804517302953.

[64] J.E. Ibarra-Esquer, F. González-Navarro, B. Flores-Rios, L. Burtseva, and M. Astorga-Vargas. Tracking the evolution of the internet of things concept across different application domains. *Sensors*, 17(6) :1379, 2017.

[65] L. Jiang, L. D. Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu. An iot-oriented data storage framework in cloud computing platform. *IEEE Transactions on Industrial Informatics*, 10(2) :1443–1451, May 2014. ISSN 1551-3203. doi : 10.1109/TII.2014.2306384.

[66] R. Khan, S.U. Khan, R. Zaheer, and S. Khan. Future internet : The internet of things architecture, possible applications and key challenges. In *2012 10th International Conference on Frontiers of Information Technology*, pages 257–260, Dec 2012. doi : 10.1109/FIT.2012.53.

[67] P. Lacomme, C. Prins, and M. Sevaux. *Algorithmes de graphes*, volume 28. Eyrolles Paris, 2003.

[68] T. López, Damith Chinthana Ranasinghe, Bela Patkai, and Duncan McFarlane. Taxonomy, technology and applications of smart objects. *Information Systems Frontiers*, 13(2) :281–300, Apr 2011. ISSN 1572-9419. doi : 10.1007/s10796-009-9218-4. URL https://doi.org/10.1007/s10796-009-9218-4.

[69] T.H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun. Fog computing : Focusing on mobile users at the edge. *arXiv preprint arXiv :1502.01815*, 2015.

[70] G. Marques, N. Garcia, and N. Pombo. A survey on iot : architectures, elements, applications, qos, platforms and security concepts. In *Advances in Mobile Cloud Computing and Big Data in the 5G Era*, pages 115–130. Springer, 2017.

[71] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran. Emufog : Extensible and scalable emulation of large-scale fog computing infrastructures. 2017.

[72] A. Mebrek, L. Merghem-Boulahia, and M. Esseghir. Efficient green solution for a balanced energy consumption and delay in the iot-fog-cloud computing. In *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pages 1–4, Oct 2017. doi : 10.1109/NCA.2017.8171359.

[73] B. Menegola. A study of the k-way graph partitioning problem. 2012.

[74] P. Meye, P. Raipin, F. Tronel, and E. Anceaume. Mistore : A distributed storage system leveraging the dsl infrastructure of an isp. In *2014 International Conference on High Performance Computing Simulation (HPCS)*, pages 260–267, July 2014. doi : 10.1109/HPCSim.2014.6903694.

[75] Q.T. Minh, C.M. Tran, T.A. Le, B.T. Nguyen, T.M. Tran, and R.K. Balan. Fogfly : A traffic light optimization solution based on fog computing. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, UbiComp '18, pages 1130–1139, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5966-5. doi : 10.1145/3267305.3274169. URL http://doi.acm.org/10.1145/3267305.3274169.

[76] M.I. Naas, P. Raipin, J. Boukhobza, and L. Lemarchand. iFogStor : an IoT Data Placement Strategy for Fog Infrastructure. In *IEEE 1st International Conference on Fog and Edge Computing*, Madrid, Spain, May 2017. doi : 10.1109/ICFEC.2017.15. URL http://hal.univ-brest.fr/hal-01558220.

[77] M.I. Naas, J. Boukhobza, P. Raipin, and L. Lemarchand. An extension to ifogsim to enable the design of data placement strategies. In *2nd IEEE International Conference on Fog and Edge Computing, ICFEC 2018, Washington DC, USA, May 1-3, 2018*, pages 1–8, 2018. doi : 10.1109/CFEC.2018.8358724. URL https://doi.org/10.1109/CFEC.2018.8358724.

[78] M.I. Naas, L. Lemarchand, J. Boukhobza, and P. Raipin. A graph partitioning-based heuristic for runtime iot data placement strategies in a fog infrastructure. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*, pages 767–774,

2018. doi : 10.1145/3167132.3167217. URL http://doi.acm.org/10.1145/3167132.3167217.

[79] M. Ouabiba, N. Mebarki, and P. Castagna. Couplage entre des méthodes d'optimisation itératives et des modèles de simulation à évenements discrets. In *3 ème Conférence Francophone de MOdélisation et SIMulation «Conception, Analyse et Gestion des Systèmes Industriels» MOSIM'01–du 25 au 27 avril 2001-Troyes (France)*, 2001.

[80] T.V. Rao, A. Khan, M. Maschendra, and M.K. Kumar. A paradigm shift from cloud to fog computing. *International Journal of Science, Engineering and Computer Technology*, 5(11) :385, 2015.

[81] H.U. Rehman, M. Asif, and M. Ahmad. Future applications and research challenges of iot. In *2017 International Conference on Information and Communication Technologies (ICICT)*, pages 68–74, Dec 2017. doi : 10.1109/ICICT.2017.8320166.

[82] K.P. Saharan and An. Kumar. Fog in comparison to cloud : A survey. *International Journal of Computer Applications*, 122(3) :10–12, July 2015.

[83] S. Sarkar, S. Chatterjee, and S. Misra. Assessment of the suitability of fog computing in the context of internet of things. *IEEE Transactions on Cloud Computing*, PP(99) :46–59, 2015. ISSN 2168-7161. doi : 10.1109/TCC.2015.2485206.

[84] P. Sethi and S. Sarangi. Internet of things : architectures, protocols, and applications. *Journal of Electrical and Computer Engineering*, 2017, 2017.

[85] Y. Shi, G. Ding, H. Wang, H.E. Roman, and S. Lu. The fog computing service for healthcare. In *Future Information and Communication Technologies for Ubiquitous HealthCare (Ubi-HealthTech), 2015 2nd International Symposium on*, pages 1–5. IEEE, 2015.

[86] Y. Simmhan. Big data and fog computing. *arXiv preprint arXiv :1712.09552*, 2017.

[87] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner. Resource provisioning for iot services in the fog. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 32–39, Nov 2016. doi : 10.1109/SOCA.2016.10.

[88] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar. Towards qos-aware fog service placement. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 89–96, May 2017. doi : 10.1109/ICFEC.2017.12.

[89] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya. A toolkit for modelling and simulating data grids : An extension to gridsim. *Concurr. Comput. : Pract. Exper.*, 20(13) :1591–1609, 2008. ISSN 1532-0626. doi : 10.1002/cpe.v20:13. URL http://dx.doi.org/10.1002/cpe.v20:13.

[90] M. Taneja and A. Davy. Resource aware placement of iot application modules in fog-cloud computing paradigm. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1222–1228, May 2017. doi : 10.23919/INM.2017.7987464.

[91] R.L. Rivest T.H. Cormen, C.E. Leiserson and C. Stein. Introduction to algorithms. pages 693–700, 2009.

[92] V. Turner, J. Gantz, D. Reinsel, and S. Minton. The digital universe of opportunities : Rich data and the increasing value of the internet of things. *IDC Analyze the Future*, 16, 2014.

[93] A. Vahid Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya. Fog Computing : Principles, Architectures, and Applications. *ArXiv e-prints*, January 2016.

[94] K. Velasquez, D. Abreu, M. Curado, and E. Monteiro. Service placement for latency reduction in the internet of things. *Annals of Telecommunications*, 72(1) :105–115, 2017. ISSN 1958-9395. doi : 10.1007/s12243-016-0524-9. URL https://doi.org/10.1007/s12243-016-0524-9.

[95] Y. Xia, X. Etchevers, L. Letondeur, T. Coupaye, and F. Desprez. Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed iot applications in the fog. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 751–760, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5191-1. doi : 10.1145/3167132.3167215. URL http://doi.acm.org/10.1145/3167132.3167215.

[96] D. Yang, F. Liu, and Y. Liang. A survey of the internet of things. In *Proceedings of the 1st International Conference on E-Business Intelligence (ICEBI2010),*. Atlantis Press, 2010.

[97] Z. Yang, Y. Yue, Y. Yang, Y. Peng, X. Wang, and W. Liu. Study and application on the architecture and key technologies for iot. In *2011 International Conference on Multimedia Technology*, pages 747–751, July 2011. doi : 10.1109/ICMT.2011.6002149.

[98] S. Yi, Z. Hao, Z. Qin, and Q. Li. Fog computing : Platform and applications. In *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*, pages 73–78. IEEE, 2015.

[99] S. Yi, C. Li, and Q. Li. A survey of fog computing : concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data*, pages 37–42. ACM, 2015.

[100] A. Yousefpour, G. Ishigaki, and J.P. Jue. Fog computing : Towards minimizing delay in the internet of things. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 17–24, June 2017. doi : 10.1109/IEEE.EDGE.2017.12.

[101] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, and J. P. Jue. QoS-aware Dynamic Fog Service Provisioning. *ArXiv e-prints*, February 2018.

[102] B. Zhang, N. Mor, J. Kolb, D.S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz. The cloud is not enough : Saving iot from the cloud. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, Santa Clara, CA, 2015. USENIX Association. URL https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/zhang.

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UBO
Université de Bretagne Occidentale

Titre : Placement des données de l'Internet des Objets dans une infrastructure de Fog.

Mots clés : Internet des Objets, Fog computing, Placement de données, Optimisation, Latence réseau.

Résumé : Dans les prochaines années, l'Internet des objets (IoT) constituera l'une des applications générant le plus de données. Actuellement, les données de l'IoT sont stockées dans le Cloud. Avec l'augmentation du nombre d'objets connectés, la transmission de la grande quantité de données produite vers le Cloud génèrera des goulets d'étranglement. Par conséquent, les latences seront élevées. Afin de réduire ces latences, le Fog computing a été proposé comme un paradigme étendant les services du Cloud jusqu'aux périphéries du réseau. Il consiste à utiliser tout équipement localisé dans le réseau (ex. routeur) pour faire le stockage et le traitement des données. Cependant, le Fog présente une infrastructure hétérogène. En effet, ses équipements présentent des différences de performances de calcul, de capacités de stockage et d'interconnexions réseaux. Cette hétérogénéité peut davantage augmenter la latence du service. Cela pose un problème : le mauvais choix des emplacements de stockage des données peut augmenter la latence du service. Dans cette thèse, nous proposons une solution à ce problème sous la forme de quatre contributions : 1. Une formulation du problème de placement de données de l'IoT dans le Fog comme un programme linéaire. 2. Une solution exacte pour résoudre le problème de placement de données en utilisant CPLEX, un solveur de problème linéaire. 3. Deux heuristiques basées sur le principe de "diviser pour régner" afin de réduire le temps du calcul de placement. 4. Une plate-forme expérimentale pour évaluer des solutions de placement de données de l'IoT dans le Fog, en intégrant la gestion du placement de données à iFogSim, un simulateur d'environnement Fog et IoT.

Title: Placement of Internet of Things data in a Fog infrastructure.

Keywords: Internet of Things, Fog computing, Data placement, Optimization, Network Latency.

Abstract: In the coming years, Internet of Things (IoT) will be one of the applications generating the most data. Nowadays, IoT data is stored in the Cloud. As the number of connected objects increases, transmitting the large amount of produced data to the Cloud will create bottlenecks. As a result, latencies will be high and unpredictable. In order to reduce these latencies, Fog computing has been proposed as a paradigm extending Cloud services to the edge of the network. It consists of using any equipment located in the network (e.g. router) to store and process data. Therefore, the Fog presents a heterogeneous infrastructure. Indeed, its components have differences in computing performance, storage capacity and network interconnections. This heterogeneity can further increase the latency of the service. This raises a problem: the wrong choice of data storage locations can increase the latency of the service. In this thesis, we propose a solution to this problem in the form of four contributions: 1. A formulation of the IoT data placement problem in the Fog as a linear program. 2. An exact solution to solve the data placement problem using the CPLEX, a mixed linear problem solver. 3. Two heuristics based on the principle of "divide and conquer" to reduce the time of placement computation. 4. An experimental platform for testing and evaluating solutions for IoT data placement in the Fog, integrating data placement management with iFogSim, a Fog and IoT environment simulator.