République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie HOUARI BOUMÉDIÈNE

Faculté d'Électronique et d'Informatique
Département d'Informatique

MÉMOIRE DE LICENCE

Domaine : **Mathématiques et Informatique**
Filière : **Informatique**
Spécialité : **Réseaux et Systèmes Distribués**

Intitulé

Conception et implémentation d'une technique d'allocation de ressources dans un environnement *Fog Computing*

Réalisé par
Ayoub HAMMAL
et
Mehdi LERARI

Soutenu le : **XX XXX 2021**

Devant le jury composé de

XXXX YYYY,	Président du jury
XXXX YYYY,	Membre du jury

Dirigé par : **Dr. Khaled ZERAOULIA**

Binôme N° XXXX

Table des matières

Table des figures **3**

Glossaire **4**

1 Introduction générale **5**

2 Généralités **6**
2.1 Introduction . . . 6
2.2 Internet des objets (IoT) . . . 6
2.2.1 Définition . . . 6
2.2.2 Histoire et Évolution . . . 6
2.2.3 Architecture . . . 7
2.2.4 Quelques domaines d'application . . . 7
2.2.5 Problèmes et difficultés . . . 7
2.3 Fog Computing . . . 8
2.3.1 Histoire et évolution . . . 8
2.3.2 Définition et concept . . . 8
2.3.3 Architecture . . . 9
2.3.4 Avantages . . . 9
2.3.5 Défis et verrous scientifiques . . . 9
2.4 Migration dans les environnements Fog Computing . . . 10
2.4.1 Définitions et concepts . . . 10
2.4.2 Migration VMs-Conteneurs . . . 10
2.4.3 Avantages et inconvénients . . . 11
2.4.4 Techniques de migration . . . 12
2.4.5 Taxonomie, Travaux réalisés . . . 15
2.5 Gestion des ressources dans les environnements Fog Computing . . . 15
2.5.1 Définition . . . 16
2.5.2 Dimensions du problème . . . 16
2.5.3 Architectures proposées . . . 18
2.6 Conclusion . . . 19

3 Solution d'allocation de ressources dans un environnement *Fog Computing* : Stable Matching based Resources Allocation (SMRA) **20**
3.1 Introduction . . . 20
3.2 Motivation . . . 20
3.3 Problématique et article de référence . . . 20
3.4 Contribution . . . 21
3.4.1 Architecture SMRA . . . 21
3.4.2 Clusterisation (Clusterization) . . . 21
3.4.3 Profilage (Profiling) . . . 23
3.4.4 Scénario . . . 24

3.4.5 Description algorithmique du scénario 25
3.5 Conclusion 28

4 Implémentation et résultats **29**
4.1 Introduction 29
4.2 Outil de développement 29
4.2.1 Langage JAVA 29
4.2.2 L'outil IFogSim 29
4.2.3 Architecture de l'IFogSim 30
4.3 Développement de la conception 32
4.4 Résultats et évaluation des performances : 33
4.5 Conclusion 39

5 Conclusion générale **40**

6 Annexe **41**

Bibliographie **42**

Table des figures

2.1 Architecture de machine virtuelle, et de conteneur [1] 11
2.2 La migration à froid [2] 12
2.3 Itérations de la migration avec pré-copie [3] 13
2.4 La migration avec pré-copie [2] 13
2.5 Envoi des pages défectueuses [3] 14
2.6 La migration avec post-copie [2] 14
2.7 La migration hybride [2] 15
2.8 Organigramme qui illustre les domaines de gestion de ressource ainsi que leurs approches 16

3.1 Schémas représentant une architecture Fog traditionnelle 22
3.2 Schéma représentant une architecture *Fog* répartie en clusters 23
3.3 Schéma représentant les profils des noeuds au sein d'un cluster 24
3.4 Organigramme du scénario d'exécution du SMRA 25

4.1 Diagramme représentant l'interaction des différents composants de l'IFogSim 31
4.2 Diagramme représentant les relations entre les principales classes de l'IFogSim 32
4.3 Délai moyen d'exécution d'un tuple en fonction du nombre de niveaux 34
4.4 Délai moyen d'exécution d'une application en fonction du nombre de niveaux 35
4.5 Délai moyen d'exécution d'un tuple en fonction du nombre de nœuds par niveaux 36
4.6 Délai moyen d'exécution d'une application en fonction du nombre de nœuds par niveau . 37
4.7 Délai moyen d'exécution d'un tuple en fonction du taux de transmission 38
4.8 Délai moyen d'exécution d'une application en fonction du taux de transmission 39

Glossaire

Cloud Computing Correspond à l'accès à des services informatiques via Internet à partir d'un fournisseur de services 8

Conteneur Enveloppe virtuelle qui permet de distribuer une application avec tous les éléments dont elle a besoin pour fonctionner : fichiers source, environnement d'exécution, librairies, outils et fichiers 10

Fog Computing Technologie Cloud dans laquelle les données générées par les terminaux ne sont pas directement téléchargées dans le cloud, mais sont au préalable prétraitées dans des mini-centres de calcul décentralisés 6

IoT signifie Internet of Things (ou Internet des objets) 6

Latence Temps nécessaire à un paquet de données pour passer de la source à la destination à travers un réseau 8

Machine Virtuelle Environnement virtuel qui fonctionne comme un système informatique virtuel, avec son propre processeur, sa mémoire, son interface réseau et son espace de stockage, mais qui est créé sur un système matériel physique 10

Migration Processus qui consiste à déplacer l'état d'une instance virtuelle, d'un hôte physique à un autre 10

Virtualisation Technologie qui permet de créer plusieurs environnements simulés ou ressources dédiées à partir d'un seul système physique 10

Chapitre 1

Introduction générale

L'Internet des objets est un sujet émergent d'importance technique, sociale et économique. Les produits de consommation, les biens durables, les voitures et les camions, les composants industriels et utilitaires, les capteurs et autres objets du quotidien sont combinés à la connectivité Internet et à de puissantes capacités d'analyse de données qui promettent de transformer la façon dont nous travaillons, vivons et jouons. Les projections de l'impact de l'*IoT* sur Internet et l'économie sont impressionnantes, certains prévoyant jusqu'à 100 milliards d'appareils connectés IoT et un impact économique mondial de plus de 11 milliards de dollars d'ici 2025.

Nous proposons dans ce travail une nouvelle technique d'allocation de ressources dans un environnement *Fog/Edge Computing* visant à servir de manière optimale les demandes de services générées par un ensemble d'objets *IoT*. Nous exploitons dans cette technique l'algorithme de correspondance de Gale-Shapley prouvé stable et efficace.

Nous commençons d'abord par définir les concepts clés de la problématique et établir l'état de l'art concernant ce domaine de recherche. Puis dans un second chapitre, nous présentons notre technique, le scénario d'exécution et les différents échanges entre les entités de notre environnement. Enfin, nous proposons dans le dernier chapitre une implémentation de cette technique dans un environnement de simulation, ainsi qu'une discussion et une analyse des résultats obtenus.

Chapitre 2

Généralités

2.1 Introduction

Nous présentons dans ce chapitre les principaux concepts liés au *Fog Computing*. Nous commençons d'abord par introduire l'Internet des objets et leur relation au Fog Computing. Puis, nous expliquons les techniques de migration de conteneurs entre les nœuds Fog. Enfin, nous terminons ce chapitre par une taxonomie des travaux similaires axés sur la gestion de ressources des nœuds Fog, ainsi que les différents défis rencontrés.

2.2 Internet des objets (IoT)

2.2.1 Définition

Le terme Internet des objets (Internet of Things ou *IoT*) décrit le réseau d'appareils physiques - souvent hétérogène - interconnectés, et dont le rôle principal est la récolte et l'échange d'information, ainsi que l'interaction avec l'environnement extérieur [4]. Cette infrastructure est dite intelligente, dotée de la capacité de s'auto-organiser, partager l'information de manière optimale et de réagir aux changements environnementaux [5].

Les objets dans ce type de réseau sont de capacité limitée, que ce soit en puissance de calcul ou en consommation d'énergie. Ce sont majoritairement des objets électroniques quotidiens (à l'exemple de smartphones, véhicules ou équipements ménagers), chacun avec sa propre identité pour communiquer avec le reste des objets et synchroniser les efforts de réponses aux différentes situations externes auxquelles ils sont exposés.

2.2.2 Histoire et Évolution

L'idée d'interconnecter des dispositifs électroniques moyennant un réseau informatique est apparue dans les années 80s, à l'Université de Carnegie Melon, où on avait relié un distributeur de boissons fraîches à un ordinateur de monitorage. Ce qui a permis aux programmeurs de consulter la disponibilité des boissons à distance et d'éviter les trajets inutiles [5].

Cependant, le premier à avoir introduit ce concept est Kevin Ashton en 1999, qui travaillait dans l'optimisation de chaîne de production chez Procter & Gamble. Ashton avait constaté qu'en 1999, 50 petabytes d'information étaient créés par des êtres humains. D'où, il a dévelopé l'idée de décharger l'homme de cette tâche fastidieuse de récolte d'information et ce, lors d'une présentation portant le nom de la technologie (Internet of Things). Il propose plutôt d'exploiter la masse d'engins et de capteurs disponibles et déjà déployés à ce moment là [6].

Cette infrastructure n'a été adoptée en industrie qu'en fin de l'année 2000s. Depuis, les constructeurs du domaine de téléinformatique concentrent leurs efforts sur la production de capteurs et d'appareils IoT avec différentes fonctionnalités dans le but de combler les besoins des diverses industries, allant de l'agriculture et domaine médical jusqu'à l'industrie militaire.

2.2.3 Architecture

L'architecture d'un réseau IoT se décompose en 4 couches flexibles. Chaque couche est constituée de plusieurs technologies et standards [4]. Cet aspect modulaire permet une meilleure scalabilité de l'infrastructure et une meilleure adaptation aux besoins émergeants. Les couches de ce modèle sont comme suit :

1. *La couche de capteurs et d'objets intelligents :* Elle est formée d'objets connectés munis de capteurs et/ou d'actionneurs. C'est la couche la plus proche de l'environnement physique, celle-ci transforme les événements générés par ce dernier en un flux d'information à temps réel, et se charge de leur transmission. Les capteurs ont différentes spécifications, comme la mesure de la température, ou la pression, la capture de mouvement ..., et sont connectés soit à des passerelles à l'aide de réseaux filaires (Ethernet) ou non (Wi-Fi, Bluetooth, RFID ...), soit directement à la couche applicative. Un exemple de ce type de technologie est les WSNs, caractérisés par leur basse consommation d'énergie et grande zone de couverture.
2. *La couche réseau et passerelles :* Cette couche garantit la transmission de la masse d'information générée par la couche précédente, tout en respectant la qualité de service exigée par les applications servies. Plusieurs infrastructures et protocoles de communication ont été mis en place dans le but d'optimiser l'acheminement et traitement d'information, comme le concept de Fog Computing qu'on détaillera par la suite.
3. *La couche de gestion :* Le rôle de cette couche est de filtrer et organiser les informations, en fournissant une couche d'abstraction à l'application. Elle s'occupe de la gestion de priorité, et l'analyse de la pertinence des données. C'est aussi à ce niveau que les politiques d'anonymisation et sécurisation de données sont implémentées.
4. *La couche d'application :* Elle est située majoritairement dans des clouds ou data-centers. Les applications couvrent des domaines différents comme l'agriculture et la gestion de villes intelligentes, et d'autres plus critiques, comme le domaine de la santé ou le domaine militaire. Toutefois, depuis quelques années, les efforts de recherche visent à rapprocher ces applications de la première couche du modèle, pour des motifs que nous expliquerons dans la section dédiée au Fog Computing.

2.2.4 Quelques domaines d'application

L'installation d'objets intelligents s'est démocratisée depuis quelques années. On les retrouve dans les environnements suivants :

— Les systèmes de sécurité et surveillance, où des caméras et capteurs de mouvements permettent de détecter et identifier toute activité suspecte.
— Les maisons intelligentes : ce qui décrit la connectivité des objets dans un domicile, tous œuvrent pour fournir de meilleures conditions de vie, et ceci en assurant : la régulation de températures, l'optimisation de consommation d'énergie, la détection d'incendie et le filtrage de l'air.
— Les systèmes de voitures autonomes dans lesquels les véhicules utilisant la voie publique peuvent s'échanger des informations, parfois critiques, et alerter les conducteurs de tout danger imminent. Ils peuvent aussi, en se basant sur les informations de géolocalisation fournies par les autres usagers, produire des recommandations de destination tout en évitant les points de congestion de la circulation routière.
— Les dispositifs permettant la surveillance médicale de personnes incapacités, et ainsi la prise de dispositions nécessaires dans les moments d'urgence.

2.2.5 Problèmes et difficultés

Le provisionnement en énergie est devenu une des préoccupations de la société actuelle. On cherche à optimiser l'utilisation de ressources énergétiques, soit pour étendre l'autonomie des objets et capteurs mobiles, ou bien pour réduire les frais d'approvisionnement en électricité. Des algorithmes de gestion et

allocation des ressources ont été proposés pour minimiser cette consommation, mais les recherches sur ce sujet sont toujours actives.

Avec la croissance du nombre d'objets connectés, les attaques cybercriminelles deviennent excentriques. Ce réseau peut être vulnérable contre l'injection de données erronées qui peuvent influencer des prises de décision parfois critiques. Les nœuds intercommunicants disposent de ressources limitées, et ainsi, ils peuvent être sujets à des attaques de déni de service (*DDOS*). La grande quantité de données produite doit être protégée tout au long du circuit liant les couches présentées précédemment. Un travail de supervision doit être mené dans le but de garantir la confidentialité des données récoltées, et empêcher les pratiques abusives comme la vente de données d'utilisateurs [7].

De plus, certaines applications demandent une certaine réactivité et une grande vitesse de réponse pour accomplir des missions critiques comme pour la conduite de véhicules automobiles ou la surveillance médicale. Ces exigences sont souvent restreintes par d'autres contraintes comme la mobilité des objets connectés ou la congestion dans le réseau de communication.

2.3 Fog Computing

2.3.1 Histoire et évolution

Malgré les avantages considérables que procure le *Cloud Computing* en termes de performance, d'accessibilité et de scalabilité, Ce dernier se heurte néanmoins à quelques limites face à l'expansion de l'Internet des objets (IoT), ce qui impose donc une réévaluation de ce paradigme.

En effet, l'accélération de la croissance du nombre d'appareils connectés à Internet ainsi que l'énorme quantité de données générées par ces derniers ont démontré une certaine limitation du paradigme Cloud. Le problème le plus apparent est le problème de *Latence*, en vue donc de la croissance de la quantité de données générées par les différents objets connectés et la dépendance de ces derniers visés à vis du Cloud pour le traitement de ces données, ceci risque de provoquer une réduction considérable des performances du réseau, ce qui engendrerait une augmentation des délais de transfert et donc une diminution des performances de traitement. Ce qui peut être critique au niveau de certaines applications notamment les applications qui requièrent des traitements en temps réel.
D'autres problèmes peuvent être cités également, comme le problème de congestion ou encore le problème de connaissance de localisation.

C'est pour répondre donc à ces problèmes que le Fog Computing a été proposée comme une extension du paradigme Cloud en 2012, et qui a vu par la suite la création d'un consortium dédié qu'est l'**OpenFog Consortium** afin de faciliter l'interopérabilité des différentes solutions technologiques.

2.3.2 Définition et concept

Selon CISCO [8], le terme *Fog computing* désigne : « une plate-forme hautement virtualisée qui fournit des services de calcul, de stockage et de mise en réseau entre les appareils finaux et les centres de données de Cloud Computing traditionnels, située généralement, mais pas exclusivement, à la périphérie du réseau ».

L'*OpenFog Consortium* [9] le définit aussi comme étant : « Une architecture horizontale au niveau du système qui distribue des fonctions de calcul, de stockage, de contrôle et de mise en réseau plus proches des utilisateurs le long du continuum Cloud-objet ».

Le terme réfère aussi à une infrastructure matérielle et applicative distribuée qui vise à stocker et à traiter les données issues des différents appareils connectés afin de se substituer au Cloud pour certains traitements.

La principale idée du Fog Computing est l'instrumentalisation des différents équipements qui constitue les noeuds du réseau (routeurs, commutateurs, passerelles, etc.) comme étant un centre de traitement et de stockage de donnés distribué qui est à la fois intermédiaire au Cloud et en même temps proche des extrémités du réseau. En créant donc une couche auprès de la production des données, ceci

entraîne une réduction des transferts entrant et sortant du Cloud et donc une réduction de la latence, et par conséquent du temps des différents traitements et services.

2.3.3 Architecture

La plupart des recherches qui s'orientent vers la définition d'un modèle architectural semblent se diriger vers un même modèle, à savoir un modèle en 3 couches [10].
Les couches de ce modèle sont présentées comme suit :

Couche IoT : cette couche désigne l'ensemble des appareils se trouvant à l'extrémité du réseau, elle est composée essentiellement d'appareils IoT tels que des véhicules intelligents, des smartphones, des drones militaires, des capteurs sensoriels, etc. Le rôle de ces derniers étant la collecte et la transmission des données vers la couche supérieure pour stockage ou traitement.

Couche de Fog : cette couche constitue le point central du paradigme Fog, elle est constituée d'un ensemble de nœuds Fog, qui selon le OpenFog Consortium, ce dernier se définit comme « un élément physique ou logique qui implémente les services informatiques Fog » [10].

L'ensemble des nœuds constitue un centre de traitement et de stockage distribué connecté à la fois à la couche inférieure et celle supérieure à travers des nœuds passerelles, permettant ainsi de bénéficier des différents services fournis par la couche supérieure qu'est le Cloud par exemple le stockage, tout en fournissant des informations contextuelles aux utilisateurs au niveau de la couche inférieure.

Couche cloud : cette couche représente une infrastructure Cloud centralisée. Elle est composée de ressources matérielles élevées et fournit différents services, ayant comme différences avec une architecture en Cloud classique, certains traitements et services qui sont déchargés de la couche Cloud au profit de la couche Fog afin d'équilibrer la charge de travail et d'augmenter l'efficacité et la fiabilité.

2.3.4 Avantages

Le Fog computing présente de nombreux avantages qu'il convient de souligner, Ils sont généralement résumés sous le sigle SCALE pour :

— *Sécurité :* Dans ce paradigme, la sécurité est prise en considération lors de l'élaboration de l'architecture plutôt qu'une partie optionnelle.
— *Cognition :* vient du fait que l'infrastructure Fog est consciente des besoins et exigences des utilisateurs. Elle distribue ainsi plus finement les ressources en fonction de chaque utilisateur contrairement au Cloud.
— *Agilité :* Ce qui désigne la capacité d'adaptation rapide à l'innovation.
— *Latence :* 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, évitant ainsi les problèmes résultant des systèmes centralisés.
— *Efficacité :* vient du fait que cette vision étant les capacités du Cloud en intégrant les différents nœuds qui composent le réseau à l'infrastructure de traitement et de stockage, augmentant ainsi la capacité ainsi que l'efficacité globale du système.

2.3.5 Défis et verrous scientifiques

Bien que le Fog Computing ait apporté des avantages considérables, ce paradigme reste relativement récent et nécessite d'investiguer certains défis, par exemple :

— *Gestion de l'énergie :* Les infrastructures de Fog comprennent généralement un grand nombre de nœuds géo-distribués. La consommation énergétique est donc plus élevée en comparaison avec celle du Cloud. De grands efforts de recherche sont alors nécessaires pour développer des solutions efficaces afin de minimiser l'empreinte énergétique, par exemple, des algorithmes de traitement et des protocoles de communication moins coûteux en énergie sont à développer.

— *Hétérogénéité :* En plus de l'hétérogénéité trouvée dans les environnements IoT, aux nivaux des types d'objets connectés, des données, des technologies de communications et des performances. Ce problème est également présent dans les infrastructures Fog en raison de la diversité des équipements qui constituent les nœuds Fog. La gestion de l'hétérogénéité dans un environnement de Fog et d'IoT représente un défi important.
— *Gestion et provision de ressources :* Les noeuds Fog sont généralement des équipements à capacité limités, il est par conséquent indispensable de disposer de solution efficace en termes de gestion de ressource, par exemple l'ordonnancement des différentes applications.

2.4 Migration dans les environnements Fog Computing

2.4.1 Définitions et concepts

La *Virtualisation* introduit une couche d'abstraction logicielle entre le matériel et le système d'exploitation ou les applications qui s'exécutent dessus. En séparant les ressources logiques des ressources physiques sous-adjacentes, la virtualisation permet l'affectation flexible de charges de travail entre les machines physiques. La *Migration* d'instance virtuelle (*Machine Virtuelle* ou *Conteneur*) est considérée comme le processus de copie et déplacement de l'état de cette dernière d'un hôte physique à un autre [11].

2.4.2 Migration VMs-Conteneurs

La migration d'instance virtuelle joue un rôle essentiel dans les environnements Fog étant donné qu'elle permet de garantir la continuité des services, quels que soient les besoins en mobilité exprimés par les objets connectés à cet environnement. Il existe deux grandes techniques de virtualisation logicielle exploitées dans les architectures orientées services : les machines virtuelles et les conteneurs. Leurs principales différences résident dans leur évolutivité et leur portabilité [12] (voir Figure 2.1).

Dans le cas général, le volume d'un conteneur se compte en mégaoctets. Il ne comporte rien de plus gros qu'une application et ses fichiers qu'elle en dépend. Ce sont des paquets à monofonctionnalité, effectuant des tâches spécifiques (appelées microservices) [12]. C'est une méthode de virtualisation au niveau du système d'exploitation(Software Virtualization) qui vise à exécuter plusieurs systèmes totalement isolés (conteneurs) sur un seul hôte de contrôle (un simple système d'exploitation). Les conteneurs partagent le noyau avec l'hôte et fournissent un environnement indépendant qui possède son propre CPU, mémoire, bloc d'E/S, réseau et le mécanisme de contrôle des ressources.

D'autre part, une machine virtuelle est plus volumineuse (plusieurs gigaoctets), comporte son propre système d'exploitation y compris son noyau. Elle permet l'exécution simultanée de plusieurs fonctions gourmandes en ressources [13]. L'ensemble de machines virtuelles sur une même machine sont gérées par un hyperviseur, qui permet leur isolation en s'exécutant sur des hôtes physiques et il est également responsable de la coexistence de différents noyaux des machines virtuelles au sein d'une même machine physique.

Donc la différence principale est que les conteneurs permettent de virtualiser un système d'exploitation afin que plusieurs charges de travail s'exécutent sur un système d'exploitation unique (on peut prendre l'exemple d'une base de données MySQL qu'on aura comme copie individuelle dans une machine virtuelle, mais avec l'utilisation des conteneurs on aura la possibilité d'avoir plusieurs copies des services de ce dernier).

Les conteneurs partagent en pratique le même noyau du système d'exploitation hôte contrairement aux technologies de machines virtuelles où le matériel est virtualisé pour exécuter plusieurs instances de système d'exploitation. Cela permet de lancer un nombre beaucoup plus important de conteneurs que de machines virtuelles sur le même matériel, et ainsi fournir une haute disponibilité capable de satisfaire la demande imposée par les réseaux de nouvelle génération telle que la 5G. Des exemples de systèmes à base de conteneurs sont Docker, OpenVZ, LXC et LXD.

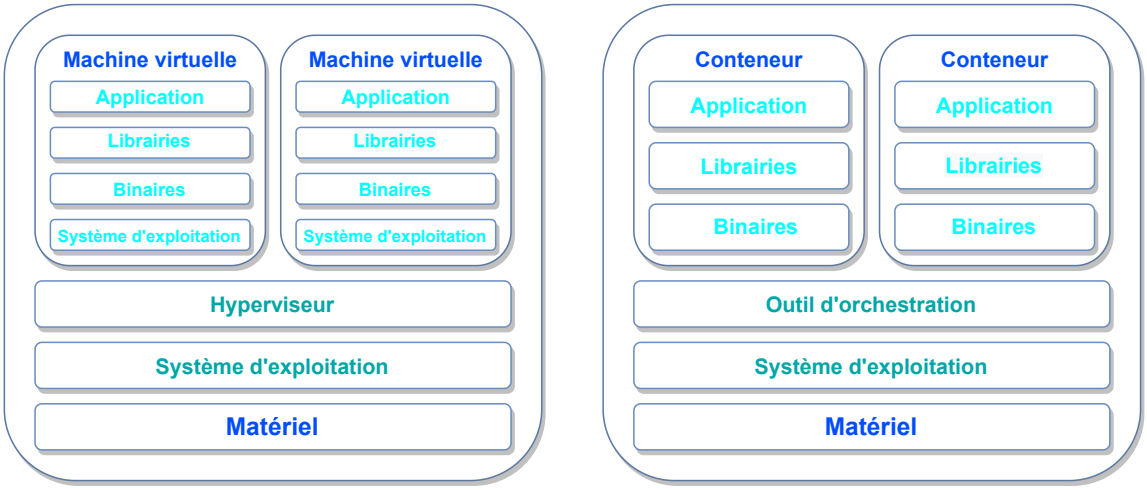

FIGURE 2.1 – Architecture de machine virtuelle, et de conteneur [1]

2.4.3 Avantages et inconvénients

Avantages de la virtualisation et de la migration

La dynamicité de ce mécanisme de migration confère une flexibilité dans la distribution et organisation des tâches de travail sur l'ensemble des ressources matérielles. Cela permet de répondre efficacement aux fluctuations des charges en mobilisant les conteneurs et machines virtuelles selon le besoin. Deux utilisations sont envisageables comme suit [11].

D'une part, la migration permet de consolider les serveurs, en regroupant les instances virtuelles sur un nombre réduit de machines. Cette utilisation est particulièrement efficace dans le scénario de charge réduite et considérablement inférieure à la capacité de traitement totale disponible. Ceci afin de permettre la mise en veille ou la suspension des machines non exploitées et en l'occurrence réduire la consommation énergétique et ainsi les coûts engendrés.

D'une autre part, et dans les conditions opposées à celles susmentionnées, c'est-à-dire dans un environnement qui exige de grandes capacités de traitement, la précédente technique cause une surcharge sur les machines exploitées et par la suite une dégradation du temps de réponse et réduction du temps de disponibilité du service. Pour pallier ces contrecoups, la migration de machines virtuelles et de conteneurs est employée pour instaurer et assurer une uniformité dans la distribution des tâches entre les ressources physiques disponibles.

Ainsi, trouver un compromis optimal entre la consolidation de serveurs et l'équilibrage de charge est essentiel pour parvenir à une utilisation efficace des ressources dans les centres de données virtualisés.

Le mécanisme de migration permet aussi d'améliorer la localisation des données dans un réseau notamment dans des environnements Fog ou pour servir des applications hautement mobiles, résultant en une meilleure exploitation de la bande passante et la réduction du délai de communication.

De plus, ce mécanisme permet de conserver la continuité du service lors des maintenances routinières ou de pannes matérielles, et de minimiser les conséquences des erreurs humaines ou catastrophes naturelles en transférant les services critiques de façon réactive.

Inconvénients

Malgré ces avantages, la migration de VM (machine virtuelle) ou de contenaire présente quelques désavantages, parmi lesquels on cite :

— Le coût en consommation de ressources engendré par l'opération de migration, en bande passante, temps de calcul CPU et disque.
— La discontinuité du service malgré l'existence de techniques qui réduisent ce dernier, mais qui reste toutefois inévitable.

— Dans le cloud public actuel, les machines virtuelles sont installées sur les mêmes machines physiques. Certaines des machines virtuelles travaillant dans le même sous-réseau ou serveur physique peuvent collaborer afin de satisfaire un service. La collaboration et les connexions entre VM via le réseau ainsi que le partage de ressources physiques augmentent le risque de vulnérabilité de sécurité, et de contamination par des VM malicieuses [14].

2.4.4 Techniques de migration

Nous nous intéresserons dans cette partie aux techniques de migration de conteneurs. Cependant, les idées de base s'appliquent également à la migration des applications et à la migration des VMs. La migration a été rendue possible grâce à l'introduction des technologies de la virtualisation, ces derniers ont permis la séparation entre la charge de travail (workload) et le matériel du serveur (hardware).

Le temps d'arrêt (Downtime) est la période pendant laquelle les services fournis par la VM ou le conteneur migrant ne sont pas disponibles ou ne répondent plus aux demandes des utilisateurs.

Le temps total de migration est la durée de temps qui sépare le lancement du processus de migration et l'instant de mise à disposition de l'instance du serveur de destination. Ce qui correspond à la somme du temps d'arrêt et temps nécessaire pour la copie du disque et la mémoire système.

Nous distinguons deux types de migrations [2] :

La migration sans état : Le conteneur est redémarré de nouveau sur le nouvel hôte ce qui implique la perte de l'ancien état d'exécution. Elle se compose de deux étapes :

1. Lancement du nouveau conteneur sur la machine de destination.
2. L'arrêt et suppression de l'ancien conteneur de la machine source.

La migration avec état : Dans ce type de migration, on conserve les données et contextes d'exécution lors du transfert, et deux techniques sont utilisées :

1. *La migration à froid* : Dans cette approche l'instance est suspendue au lieu d'être complètement arrêté comme dans une migration froide. L'environnement virtuel est déplacé vers un autre serveur et le système d'exploitation est repris à destination. L'état des applications dans le système d'exploitation invité peut être conservé pendant la migration (stateful-state). La durée de l'indisponibilité est égale à la durée totale de migration (voir Figure 2.2). Par contre, les pages mémoires ne sont transférées qu'une seule fois, ce qui réduit le temps de transfert et la quantité de données échangées.

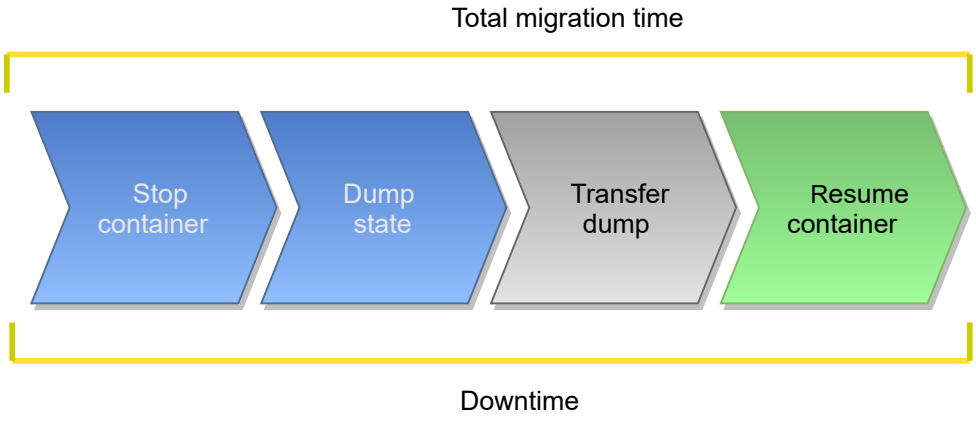

FIGURE 2.2 – La migration à froid [2]

2. *La migration à chaud* : Contrairement à la migration à froid, les pages de mémoire de l'instance seront conservées et copiées vers l'hôte de destination pendant son exécution. Après quoi il reprend son exécution sur l'hôte de destination. On distingue trois sous-catégories de migration à chaud :

(a) La migration avec pré-copie.
(b) La migration avec post-copie.
(c) La migration hybride.

La migration avec pré-copie (méthode itérative) : Dans ce cas, toutes les pages sont transférées au serveur de destination avant de geler le container. Mais quand les processus continuent leur exécution normale, les pages peuvent être modifiées et les pages transférées peuvent devenir obsolètes. C'est pourquoi les pages doivent être transférées itérativement. Sur la première étape, toutes les pages sont marquées d'un drapeau propre (clean flag) et transférées sur le serveur de destination [46]. Certaines pages peuvent être modifiées pendant ce processus, et l'indicateur propre sera supprimé dans ce cas. Sur la deuxième étape, seules les pages modifiées sont transférées vers le serveur de destination (voir Figure 2.3). La figure 2.4 montre le séquencement des opérations de cette migration.

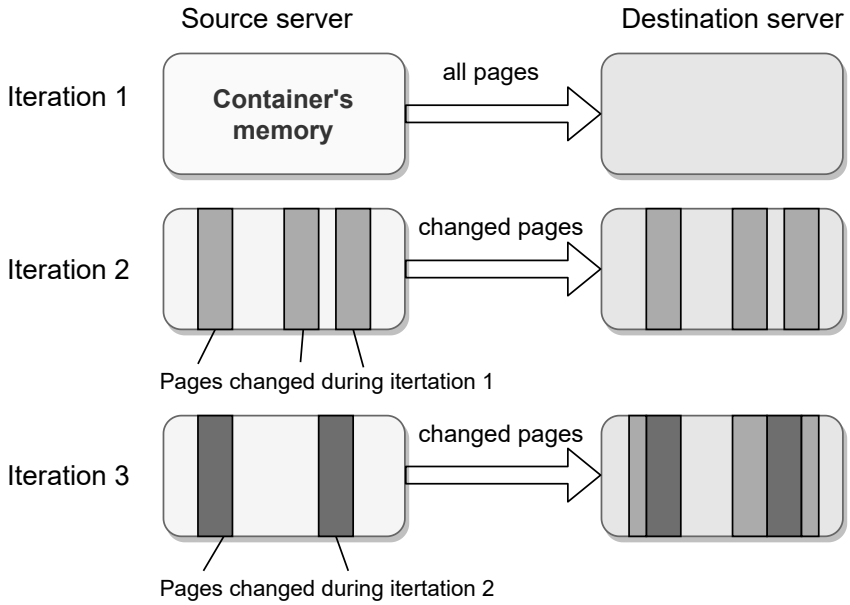

FIGURE 2.3 – Itérations de la migration avec pré-copie [3]

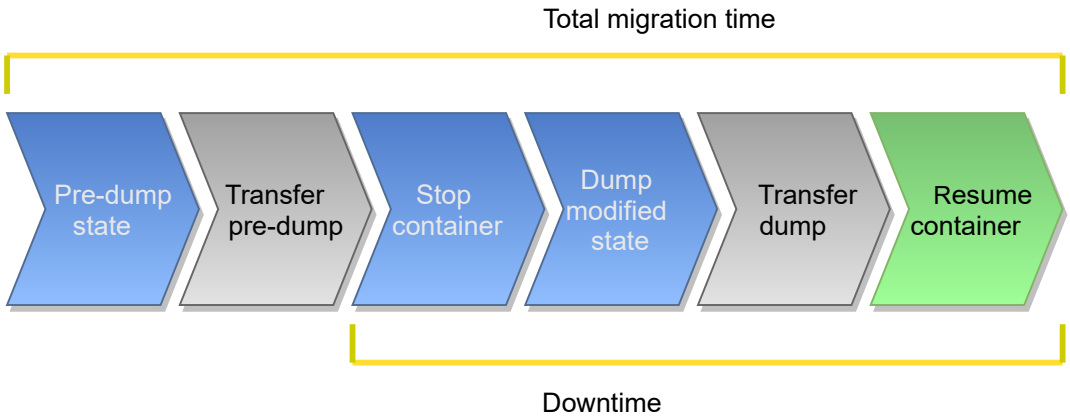

FIGURE 2.4 – La migration avec pré-copie [2]

La migration avec post-copie (fainéante) : Par opposition à la technique précédente, toutes les pages mémoire allouées par les processus sont marquées d'un drapeau spécial, qui est effacé si une page est modifiée. Après cela, un conteneur peut être gelé et son état peut être sauvegardé, mais dans ce cas, seules les pages sans cet indicateur sont stockées. Cela permet de réduire la taille d'un fichier de vidage

(dump). Puis au besoin, la machine destination génère des demandes de pages défectueuses (page-in swap), auxquelles la machine source répond en envoyant ces pages (à l'aide du daemon "page-out"), puis ces pages sont chargées en mémoire sur le serveur de destination (voir Figure 2.5). C'est pour cette raison qu'elle est appelée migration fainéante. La figure 2.6 donne le schéma global de cette migration.

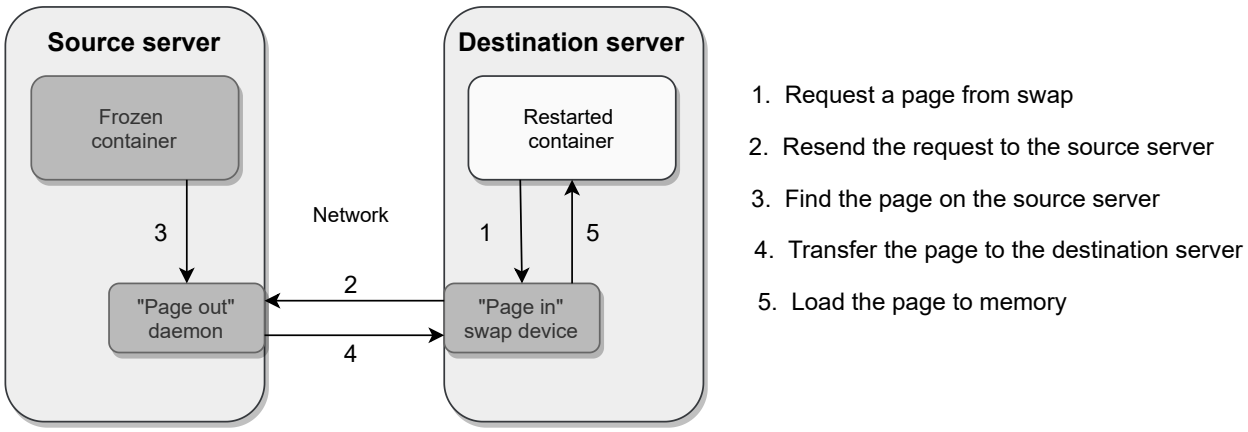

FIGURE 2.5 – Envoi des pages défectueuses [3]

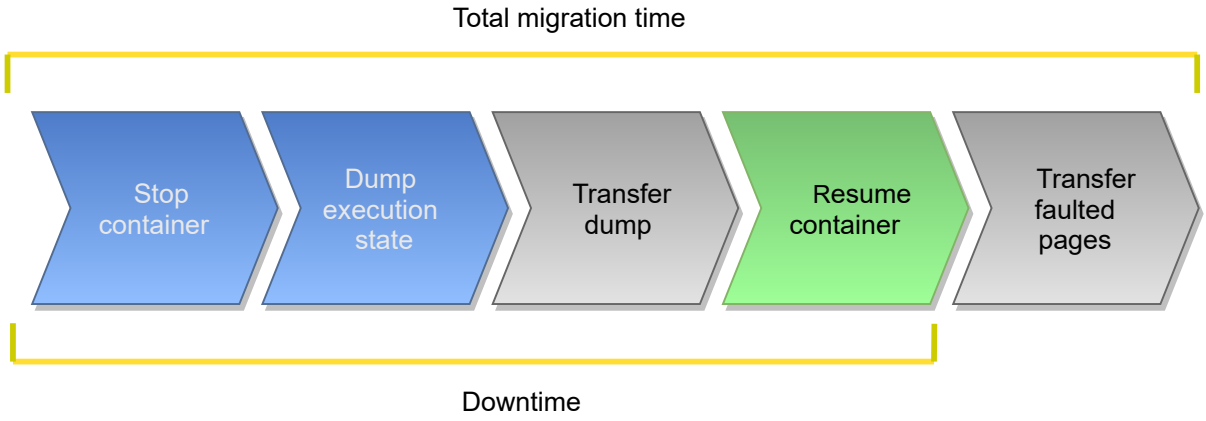

FIGURE 2.6 – La migration avec post-copie [2]

La migration hybride : Les premières étapes coïncident avec celles de la migration avec pré-copie, une seule phase de pré-copie est effectuée avant la phase de post-copie,de cette manière l'hôte cible dispose déjà de toute la mémoire, seules les pages sales devront être transférées en post-copie. Une fois la mémoire copiée, le conteneur source est arrêté, et le conteneur destination est lancé sur l'hôte cible et la phase de post-copie commence [15] (voir Figure 2.7).

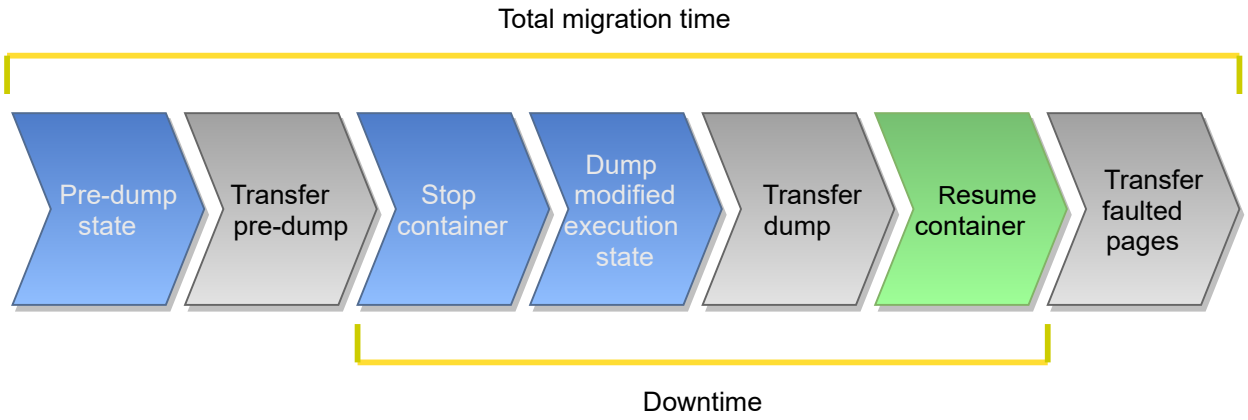

FIGURE 2.7 – La migration hybride [2]

2.4.5 Taxonomie, Travaux réalisés

Les travaux réalisés dans le domaine de virtualisation et migration de ressources virtuelles ont été classifiés selon la littérature en trois catégories selon l'objectif à optimiser [16].

— *Optimisation du coût de la migration* : Visant soit à réduire les migrations coûteuses ou à trouver des compromis de coût de migration. Les processus de décision markovien constituent l'une des approches les plus utilisées pour modéliser le compromis de migration. Plus récemment, les efforts se sont progressivement déplacés vers la résolution du problème MDP en utilisant des approches apprentissage approfondi, qui ne nécessitent pas de connaissances préalables sur la dynamique de l'environnement MDP.
— *Optimisation du temps de migration* : Les optimisations de migration focalisées sur l'axe du temps peuvent être divisées en optimisations proposées au niveau de la technologie de virtualisation et optimisations résultant de la prise d'actions proactives.
— *Optimisation du taux d'erreur de la migration* : Les recherches sur ce sujet viennent exclusivement du domaine du cloud véhiculaire, à cause de la dynamicité le caractérisant, et qui peut potentiellement troubler le taux de succès de la migration. Ici, des techniques d'intelligence artificielle sont employées pour prédire la durée de disponibilité d'un véhicule dans une zone de couverture. Des protocoles de communication plus adaptés (comme le V2V) ont été aussi proposés.

2.5 Gestion des ressources dans les environnements Fog Computing

Contrairement au *Cloud*, les ressources du *Fog* sont :

— limitées en termes de performance et d'énergie - La plupart des noeuds *Fog* possèdent généralement une puissance de calcul ainsi que des ressources énergétiques limitées, dues principalement au fait que ces derniers sont constitués généralement des équipements d'interconnexion qui composent le réseau.
— hétérogènes - aussi bien sur le plan matériel, tel que différentes architectures de processeur, que sur le plan logiciel tel que différents systèmes d'exploitation.
— sujets à des défaillances - Les nœuds *Fog* sont très susceptibles de subir des anomalies tels que des pannes de courant ou des défauts de capacité qui empêchent l'exécution des applications affectée à eux.

Dans de telles conditions, une gestion optimale des ressources est indispensable pour faire du *Fog computing* une réalité. Ce qui fait de la gestion de ressource l'un des principaux défis du paradigme.

2.5.1 Définition

Selon [17], la gestion de ressources dans les environnements *Fog* désigne « les opérations administratives telles que le déploiement, la virtualisation et la surveillance des nœuds *Fog* qui favorise les services d'infrastructure et de plate-forme basés sur le *Fog*. De plus, la gestion des ressources du *Fog* réalise l'équilibrage de charge, l'approvisionnement dynamique et la mise à l'échelle automatique pour assurer disponibilité du service et multi-location ».

2.5.2 Dimensions du problème

Le problème de la gestion des ressources dans les environnements *Fog* est un problème complexe. Il ne peut donc pas être considéré comme étant un seul problème, mais plusieurs problèmes suivant plusieurs aspects [18]. Le problème peut être vu suivant 6 axes principaux que sont : le placement d'application, l'ordonnancement des tâches, le déchargement des tâches, l'équilibrage de charges, l'allocation de ressources ainsi que l'approvisionnement en ressources.

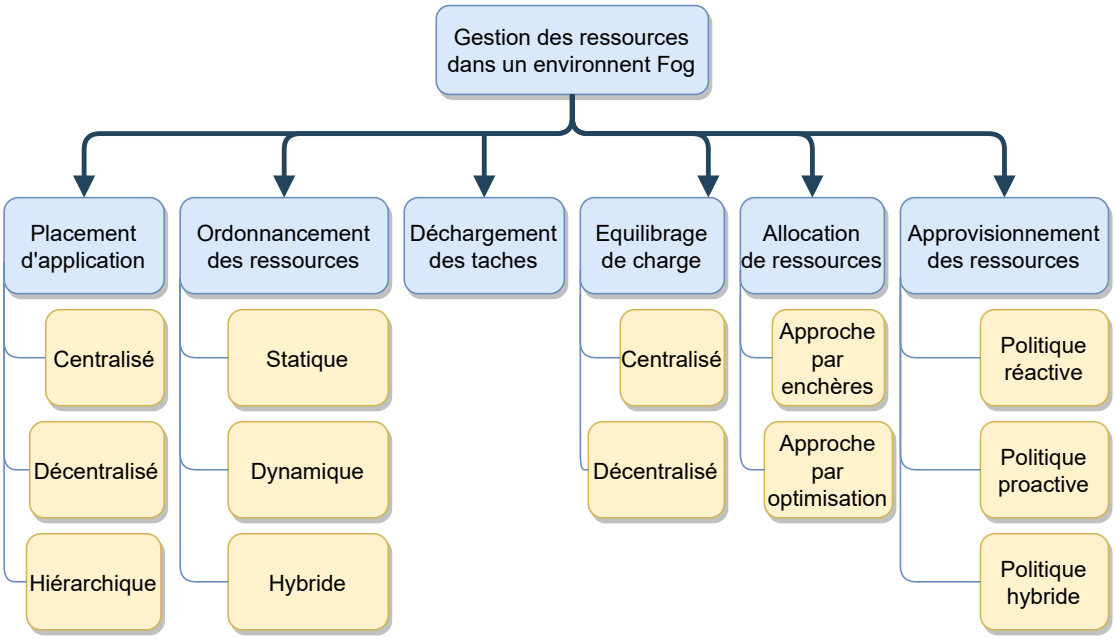

FIGURE 2.8 – Organigramme qui illustre les domaines de gestion de ressource ainsi que leurs approches

Placement d'application

Le problème de placement d'application désigne le problème de trouver une manière d'associer un service *IoT* aux nœuds *Fog* répondant aux exigences de la Qualité de service (QdS), tout en essayant de maximiser l'utilisation des différents nœuds .
D'une manière plus formelle soit S un service *IoT* avec des exigences de QdS Q, et soit N l'ensemble des noeuds *Fog*.
Une solution au problème de placement d'application consiste à associer au service S un nœud *Fog* N_i de N satisfaisant les exigences de QdS Q, tout en optimisant un ensemble de fonctions objectives O.
Il convient de préciser que les solutions peuvent être des relations multivaluées *i.e.* un service *IoT* peut être placé sur un ou plusieurs nœuds et réciproquement, un nœud peut héberger un ou plusieurs services.

Les approches basées sur la gestion de courtier "Broker" peuvent être organisées en 3 catégories que sont : l'approche centralisée, décentralisée et hiérarchique.

— Dans l'approche centralisée, le broker nécessite d'avoir une vision globale de tout l'environnement *Fog* afin de prendre des décisions d'optimisation pour l'ensemble du système. Cette ap-

proche ne garantit pas une optimisation efficace due à la difficulté d'obtenir toutes les informations de toutes les entités du *Fog*, ainsi qu'une mauvaise tolérance aux pannes dues à son architecture centralisée.
— Tandis que l'approche décentralisée, elle consiste en un ensemble d'optimisation locale ce qui la rend très intéressante en termes de scalabilité.
— Quant à l'approche hiérarchique, l'idée est de relier et de coordonner les différents gestionnaires locaux afin qu'il puisse collaborer entre eux et ainsi bénéficier des avantages des deux premières approches.

Planification des ressources :

Dans les environnements *Fog*, un service *IoT* peut être placé sur plusieurs nœuds, et chaque service peut être divisé en plusieurs sous-services.
Soit un ensemble de sous-services $S = \{S_1, ..., S_n\}$ (avec de différentes exigences en termes de QdS) placer sur un ensemble de nœuds $N = \{N_1, ..., N_m\}$ (ayant différentes capacités de traitement).
La planification de ressource consiste à trouver une affectation optimale des différents sous-services S_i aux différents nœuds N_j suivant les objectifs considérés par la politique d'ordonnancement (par exemple, minimiser le temps d'exécution).
Parmi les approches utilisées dans l'ordonnancement des ressources, les approches :
— *Statique :* Dans cette approche, l'attribution des nœuds au différent sous-service s'effectue d'une manière statique i.e, la décision est déjà prise avant même que la demande ne soit soumise, ce qui présuppose une connaissance au préalable de toutes les informations nécessaires des différentes demandes.
— *Dynamique :* contrairement à la précédente, le processus d'attribution n'est pas fixé au préalable, les décisions sont prises une fois les demandes formulées.
— *Hybride :* elle consiste en une combinaison des deux approches précédentes afin de répondre à la diversité des types d'application.

Déchargement des tâches

Le déchargement de tâches désigne le processus de transfert des tâches qui ne peuvent pas être exécutées en local pour manque de ressource vers des nœuds disposant des capacités nécessaires. Vu les ressources matérielle et énergétique limitées dont disposent les appareils *IoT*, il est souvent nécessaire de recourir à des entités externes telles que le *Fog* ou le *Cloud* afin d'exécuter des tâches gourmandes en ressource telle que des calculs graphiques, réalité augmentée, etc.
Le déchargement de tâche dépend principalement de trois composants que sont :
— *les appareils IoT :* dont le rôle est de spécifier comment les applications doivent être partitionnées, ensuite de déterminer quelle partie doit être exécutée en local, et quelle partie doit être déchargée.
— *Les liaisons de communication :* elles permettent d'assurer le transfert de tâche, et donc, la qualité des transferts dépend des capacités physiques des liaisons.
— *Les nœuds Fog :* ces derniers disposent d'une capacité plus faible que le *Cloud*, mais plus importante que les appareils *IoT*.
Le déchargement de tâche peut se produire également afin d'assurer l'équilibrage de charge, minimiser la latence, efficacité énergétique, etc.

Équilibrage de charge

L'équilibrage de charge [19] consiste à distribuer l'excédent de charges sur les différents nœuds *Fog* suivant une certaine stratégie, afin d'assurer qu'aucun nœud *Fog* ne soit en surcharge ou en sous-charge, améliorant ainsi les performances globales du système. Cependant, en réalité les mécanismes d'équilibrage de charge rencontrent de nombreux défis, principalement le problème de latence qui est due à la

migration en continu des différents processus.
Les stratégies d'équilibrage sont implémentées suivant une architecture centralisée et décentralisée.

— L'approche centralisée s'appuie sur un contrôleur central, nécessitant ainsi une connaissance globale et en temps réel de l'état des différents nœuds. Cette approche est donc difficile à implémenter dû à la difficulté de connaître en continu l'état des différents nœuds du système, mais aussi une tolérance faible aux pannes dues à son architecture centralisée.
— Quant à l'approche décentralisée, elle utilise un contrôleur décentralisé dont le rôle est de coordonner les différents contrôleurs locaux, ce qui assure une plus grande scalabilité.

Allocation des ressources

Le problème d'allocation des ressources dans les environnements *Fog* peut être considéré comme un problème de double correspondance, car les serveurs *Cloud* et les nœuds *Fog* sont couplés pour les utilisateurs et l'utilisateur et les nœuds *Fog* sont couplés pour les serveurs *Cloud*. En d'autres termes, les utilisateurs doivent prendre en considération la relation entre les nœuds *Fog* et les serveurs *Cloud*, et les serveurs *Cloud* doivent prendre en considération la relation entre les nœuds et les utilisateurs.
Les techniques d'allocation de ressource peuvent être classées en 2 principales méthodes :

— *Basé sur l'enchère "auction-based" :* les clients soumettent leurs demandes de ressources au broker avec un système de tarification des demandes, et les ressources se verront attribuer au plus offrant, en utilisant des mécanismes d'enchères calculés suivant diverses techniques mathématiques.
— *Basée sur des techniques d'optimisation :* elle consiste à trouver la combinaison optimale (serveurs-cloud,noeud-fog,utilisateur) pour chaque utilisateur en effectuant des d'optimisation de fonctions objectives, telles que la minimisation du temps de réponse, la maximisation de la QdS, etc.

Approvisionnement en ressources

Dus aux fluctuations permanentes des charges de travail des différentes applications, les problèmes de sur-approvisionnement ou sous-approvisionnement de ressources risquent de se poser.
Le problème de sur-approvisionnement consiste en une attribution d'une quantité de ressources supérieure à la charge de travail réelle d'une application. (Et réciproquement pour le problème de sous-approvisionnement).
Dans un environnement en constantes variations, un modèle statique d'approvisionnement de ressources peut être problématique, il est par conséquent indispensable d'adopter une approche dynamique permettant ainsi l'adaptation en continu vis-à- vis des charges de travail.
Les stratégies d'approvisionnement dynamique sont classées en 3 types de politique :

— *Politique réactive :* elle consiste à répondre seulement aux différentes demandes, avec aucune tentative de prédiction des prochaines demandes.
— *Politique proactive :* elle repose sur des techniques de prédiction permettant d'anticiper les prochaines évolutions des charges de travail et adapter les décisions en fonction.
— *Politique hybride :* elle adopte par conséquent les deux précédentes politiques, la politique réactive est souvent utilisée pour approvisionner des ressources à une nouvelle demande qui arrive dans le système, tandis que la politique proactive permet d'anticiper les prochaines évolutions de la demande.

2.5.3 Architectures proposées

Les différentes approches de gestion de ressources dans les environnements *Fog* ont été classées suivant leurs architectures 3 types [20] :

— *Les architectures basées sur flux de données (Data flow architectures) :* Ces types d'architecture se basent sur le sens de transfert des charges de travail, par exemple les charges de travail peuvent être transférées de l'utilisateur au nœud *Fog* ou des serveurs *Cloud* vers les nœuds.

— *Les architectures de contrôle (Control architectures) :* Ces architectures sont basées sur la manière dont les ressources sont gérées au niveau du système, par exemple un contrôleur ou un algorithme central peut être utilisé pour gérer un ensemble de nœuds.
— *L'architecture de location (Tenancy architecture) :* cette architecture se base sur la capacité des différents nœuds à héberger plusieurs applications, par exemple, une ou plusieurs applications peuvent s'exécuter sur un nœud *Fog*.

2.6 Conclusion

Nous avons présenté dans ce chapitre les différents composants logiciels et matériels de l'architectures Fog, en l'occurrence les objets IoT, les VMs et les conteneurs ainsi que les contraintes de gestion de ressources imposées par ce paradigme.

Dans ce qui suit, nous nous intéresserons à la problèmatique de la gestion efficace de ressources disponibles dans les environnements Fog. Nous introduiserons dans ce cadre notre solution d'allocation de ressources, et l'algorithme sur lequel elle se base.

Chapitre 3

Solution d'allocation de ressources dans un environnement *Fog Computing* : Stable Matching based Resources Allocation (SMRA)

3.1 Introduction

Nous avons vu à travers le chapitre précédent que le Fog Computing est caractérisé par l'hétérogénéité de ses composants à la fois matériel et applicatif. D'où le besoin d'une technique efficace permettant la gestion de l'allocation de ressources du Fog aux différentes applications l'utilisant.

Nous définirons dans ce chapitre les concepts sur lesquels repose note solution, et puis nous présenterons les algorithmes qui la composent et son scénario d'exécution nominal.

3.2 Motivation

Une bonne gestion de ressources permet d'accroître les performances globales du système, en réduisant le temps d'exécution des différentes applications, la latence, ainsi que les coûts énergétiques. Elle permet aussi d'exploiter au mieux les ressources matérielles disponibles, ce qui augmente le rendement et la rentabilité économique des différents équipements.
Par conséquent, le développement d'un bon modèle de gestion de ressource se révèle d'une importance capitale pour une exploitation efficace et rentable d'une infrastructure Fog.

3.3 Problématique et article de référence

Suite à l'étude des travaux qui traitent de l'amélioration et de l'optimisation de la gestion des ressources, on constate que le problème de planification des ressources est un problème central qui nécessite d'être investigué afin d'effectuer une gestion de ressources optimale.
Le problème consiste à concevoir un modèle de planification de ressources dynamique, efficace et évolutif, qui permet d'affecter les différentes demandes de ressources, effectuées par les appareils IoT, aux différents nœuds Fog d'une manière à optimiser au mieux certaines métriques liées aux coûts.
Pour la réalisation de ce travail, nous nous sommes inspirés d'un article publié traitant une problématique similaire [21].
Dans ce travail, les auteurs s'intéressent au processus d'approvisionnement dans un environnement *Cloud*, dans lequel ce processus peut être principalement décomposé en 3 étapes majeures à savoir :

1. L'identification des nœuds concernés (c.-à-d. les nœuds sous-utilisés ou sur-utilisés).

2. Sélection des VMs à migrer.
3. Réallocation des VMs aux nœuds sous-utilisés.

La partie qui nous intéresse est la troisième étape où ils proposent un mécanisme d'affectation de VMs aux nœuds adéquat, modélisé comme étant un problème de correspondance (matching problem).

3.4 Contribution

Dans ce travail, nous proposons une nouvelle technique d'allocation des nœuds d'un environnement Fog afin de mieux servir les requêtes générées par les objets *IoT*. Cette technique, regroupé sous le sigle *SMRA* pur *Stable Matching based Resources Allocation*, se base sur une implémentation de l'algorithme de *Gale-Shapley* [22], et sera détaillé dans ce qui suit.

Hypothèses

Afin de pouvoir nous focaliser sur le problème de correspondance entre les demandes de services et les nœuds Fog, et à cause des limites de l'environnement de simulation, nous devons d'abord formuler certaines hypothèses sur la topologie physique de l'infrastructure Fog.

— La topologie est statique durant l'exploitation : une fois la solution implémentée, la topologie ne subit pas de modification.
— La topologie suit une organisation matricielle : les nœuds Fog sont organisés en des niveaux bien distincts avec le même nombre de nœuds dans chaque niveau.
— La topologie est maillée entre les niveaux : chaque nœud Fog du cluster est relié physiquement à tous les nœuds du niveau supérieur.
— Chaque nœud à une connaissance de tous ses nœuds parents et ses nœuds enfants (c.-à-d. respectivement les nœuds du niveau supérieur et les nœuds du niveau inférieur)
— Dans ce modèle, nous disposons au minimum de trois types de requêtes qui sont :

  — La requête *demande*, qui correspond à une demande de services émise par les appareils IoT, elle comporte tous les détails de la demande.
  — la requête *résultat*, qui est généré après l'exécution d'une demande de service, elle est émise à destination de l'objet *IoT* émetteur.
  — La requête *jeton*, qui représente le jeton circulant entre les différents nœuds passerelle suivant la politique du tourniquet (round-robin).

3.4.1 Architecture SMRA

SMRA se décompose en 4 aspects interdépendants que nous détaillons par la suite :

La clusterisation (Clusterization) : en regroupant les nœuds Fog dans des groupes (Cluster) de par leur proximité géographique.

le profilage (Profiling) : en attribuant différents profiles ou rôles aux différents nœuds d'un cluster.

Le traitement par lot (Batch processing) : les demandes sont regroupées et traitées par lot selon leurs ordres d'arrivée.

La correspondance (matching) : le processus d'association et de redirection des demandes vers les nœuds Fog adéquats.

3.4.2 Clusterisation (Clusterization)

Comme vu précédemment, une infrastructure *Fog* classique est décomposée de 3 couches superposées qui sont (voir Figure 3.1) :

— *La couche IoT*, qui représente l'ensemble des objets *IoT* qui effectue des demandes de service.
— *La couche Fog*, qui représente l'ensemble des nœuds *Fog* se trouvant à l'intermédiaire entre les objets *IoT* et le *cloud*.
— *La couche Cloud* qui représente l'infrastructure Cloud traditionnelle.

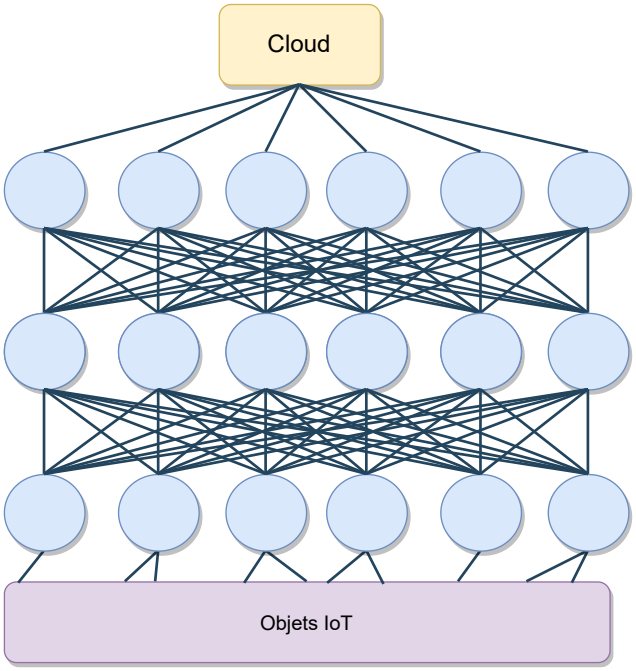

FIGURE 3.1 – Schémas représentant une architecture Fog traditionnelle

SMRA commence par découper verticalement la couche *Fog* de l'infrastructure en un ensemble de clusters [1] (voir Figure 3.2) , c.-à-d. chaque cluster regroupe un ensemble de nœuds *Fog* inter-connecté.

1. Nous discutons des critères de scalabilité verticale et horizontale du cluster dans le chapitre suivant, où nous montrons les résultats obtenus par rapport à différentes dimensions du cluster.

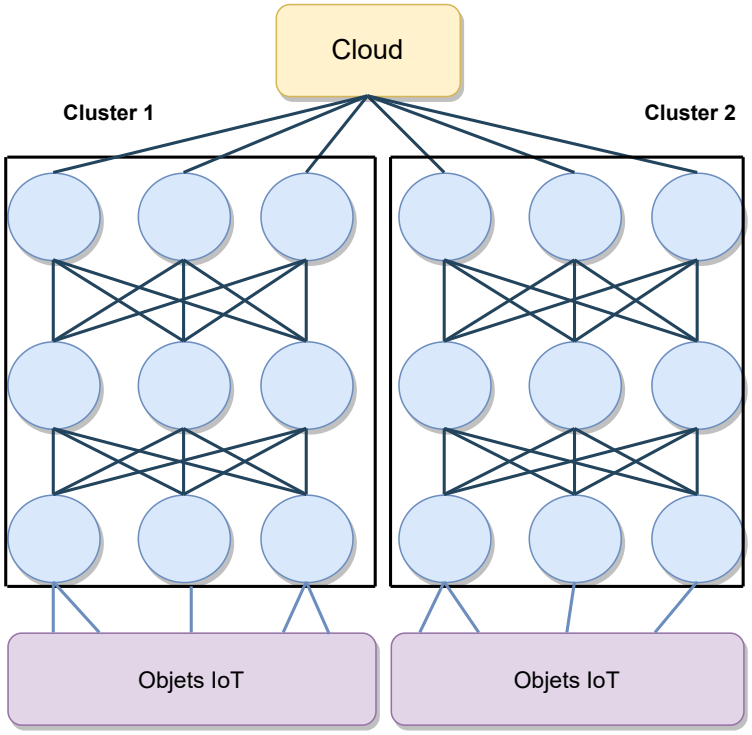

FIGURE 3.2 – Schéma représentant une architecture *Fog* répartie en clusters

3.4.3 Profilage (Profiling)

Une fois l'infrastructure est répartie en cluster, SMRA attribue des profiles aux noeuds constituant le cluster (voir Figure 3.3) en définissons deux profile que sont :

— Le profil *Noeud Passerelles* : qui est adoptée par les noeuds du niveau connecté directement à la couche *IoT*.

— Le profil *Noeud Fog* : qui est attribué aux noeuds des niveaux intermédiaires restants entre la couche des nœuds passerelles et le *cloud*.

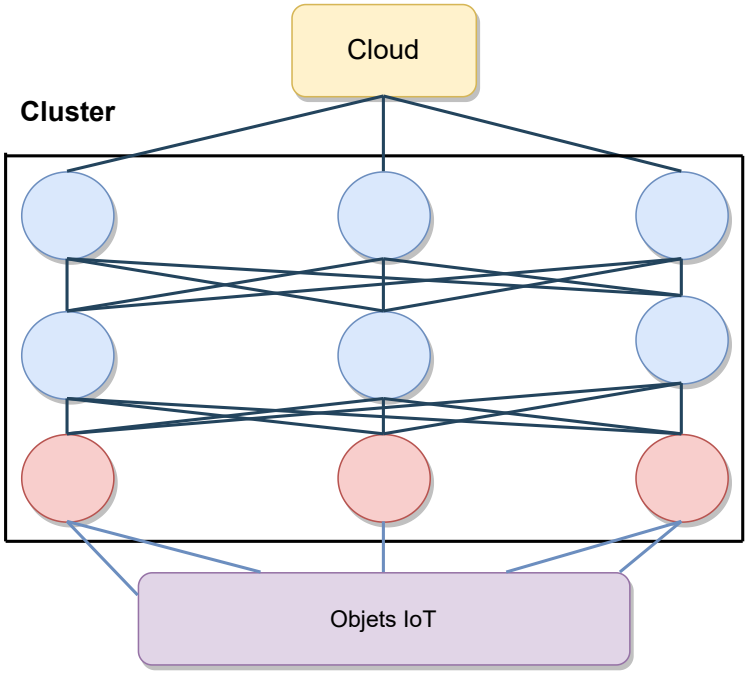

FIGURE 3.3 – Schéma représentant les profils des noeuds au sein d'un cluster

3.4.4 Scénario

Ce scénario s'applique pour un cluster, puisque les clusters sont indépendants, il suffit par la suite de généraliser ce fonctionnement à l'ensemble des clusters défini par l'infrastructure.

Les nœuds passerelles du cluster, reçoivent des demandes de services des différents appareils IoT. Chaque nœud préserve ses demandes dans une file en attendant leur correspondance puis leur envoi à leur destination.

Nous définissons un jeton circulant d'un nœud passerelle à un autre suivant la politique du tourniquet (round robin). Si un nœud passerelle possède le jeton à la réception alors il effectue une correspondance des demandes qui se trouve dans sa file d'attente avec les nœuds *Fog* du cluster, puis il envoie chaque demande au nœud *Fog* correspondant, et renvoie le jeton au prochain nœud passerelle et ainsi de suite (voir Figure 3.4).

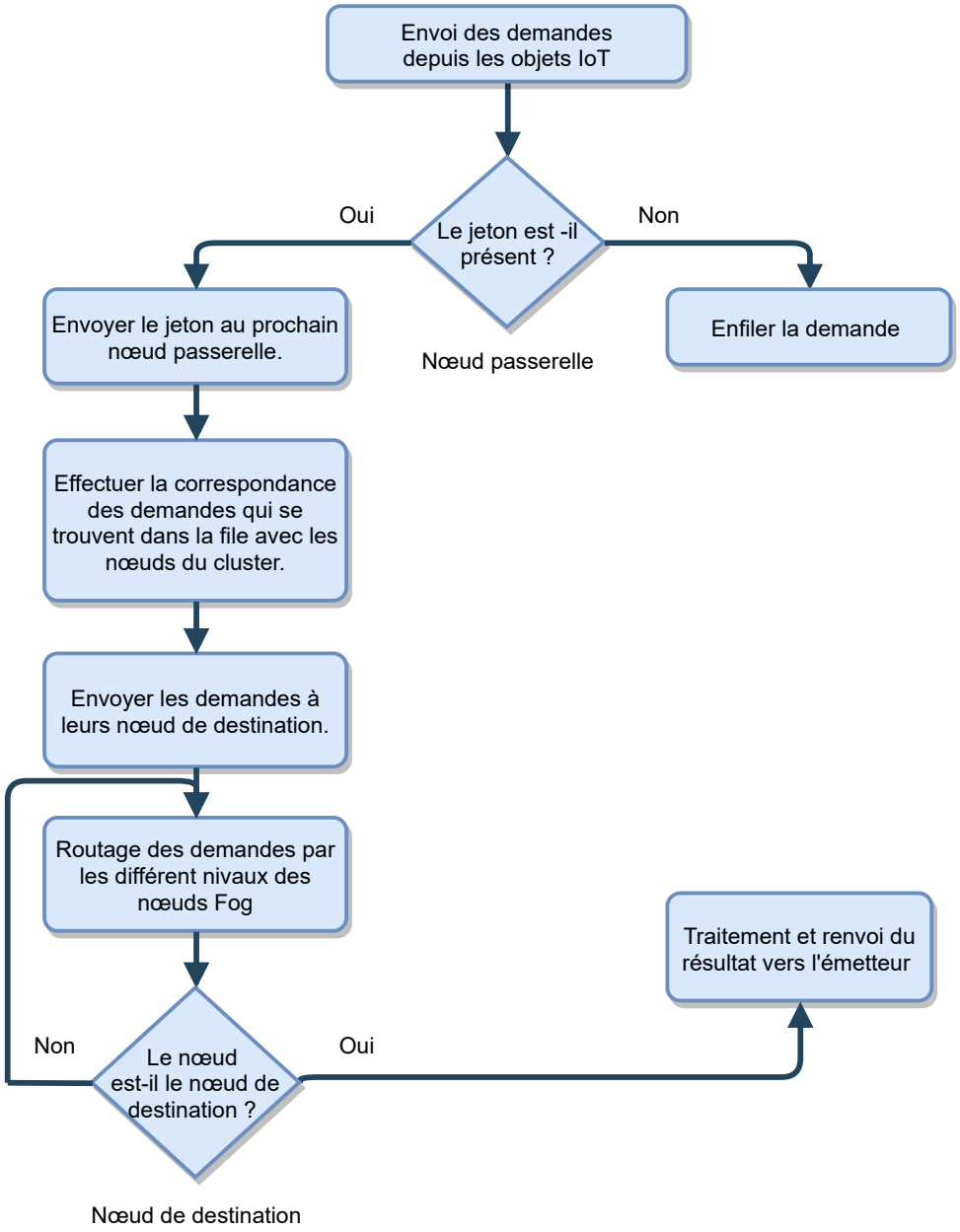

FIGURE 3.4 – Organigramme du scénario d'exécution du SMRA

3.4.5 Description algorithmique du scénario

SMRA est conçu suivant le paradigme événementiel, c.-à-d. une approche algorithmique basée sur la notion d'événement, où on cherche à associer à chaque événement une procédure à exécuter appelée "Routine". SMRA comporte 2 routines principales qui sont :

— *La Routine associée aux nœuds passerelles*
— *La Routine associée au nœud Fog*

Routine associée aux nœuds passerelles

Cette routine est exécutée au niveau des nœuds passerelles à chaque fois qu'une requête est reçue. Le pseudo-code de cette routine est le suivant :

```
Données : ListeDemandes, ListeNoeuds
si Requête reçue est de type jeton alors
    // appeler la procédure Correspondance.
    correspondance(fileDemandes, listeNoeuds) ;
    // puis on effectue le routage des demandes.
    pour chaque Demande D_i ∈ fileDemandes faire
        si listeParents.contient(D_i.destination) alors
            envoyer(demande, D_i.destination) ;
        sinon
            envoyer(demande, NoeudPèreParDéfaut) ;
            // on met à jour le nœud par défaut à chaque fois afin d'équilibrer la charge entre les
              différentes liaisons.
            NoeudPèreParDéfaut ← prochainNoeudParent() ;
        fin
    fin
sinon
    si Requête reçue est de type résultat alors
        // envoyer le résultat à l'appareil IoT concerné.
        envoyer(resultat, resultat.destination) ;
    sinon
        // la requête est par conséquent de type demande.
        enfiler(fileDemandes, Demande) ;
    fin
fin
```

Algorithme 1 : Routine associée aux nœuds passerelles

Procédure de correspondance : Pour l'implémentation de cette procédure, nous avons opté pour l'utilisation de l'algorithme de Gale-Shapley [22], qui est un algorithme conçu pour résoudre le problème des mariages stables.
Il présente des propriétés intéressantes qui sont :

- De bonne performance due à sa complexité quadratique ($\mathcal{O}(n^2)$).
- Convergence : toute demande sera associée à un nœud à la fin de l'exécution.
- Les couples (demande,nœud) résultant de cet algorithme sont stables (voir Annexe).
- La configuration résultante est optimale en comparaison à toutes les autres solutions stables [22].

Cet algorithme nécessite la définition d'une relation de préférence, que nous nommerons "Relation d'adéquation" ou RA qui est associée à chaque demande et à chaque nœud :
Pour ce faire, nous définissons une distance entre la demande et le nœud, qui est calculé de la manière suivante :

Symbole	Définition
C_u	MIPS [a] utilisé
C_{dem}	MIPS exigés par le service à exécuter
C_fd	MIPS total du nœud
C_r	MIPS disponible

$$Distance = \begin{cases} \frac{C_u + Cdem}{C_fd} / & \text{si } C_r > C_{dem} \\ -1 & \text{sinon} \end{cases}$$

a. million d'instructions par seconde

La relation d'adéquation est définie par la distance minimale entre le nœud et la demande.
Autrement dit, soit $D = \{D_1, D_2, .., D_n\}$ un ensemble de demandes, et soit $N = \{N_1, N_2, .., N_n\}$.
On dit que le nœud N_j est le mieux adéquat à la demande D_i ssi :
$Distance(D_i, N_j) = Min(Distance(D_i, N_m)), \forall m \in \{1, .., n\}$.

Pseudo-code de la procédure :

```
Données : ListeDemandes, ListeNoeuds
// initialisation de toutes les demandes à une destination null.
pour chaque Demande D_i ∈ ListeDemandes faire
    D_i.destination ← null;
fin
tant que ∃ une demande d non affectée qui peut se proposer à un nœud faire
    n ← le nœud le mieux adéquat à d telque d ∉ nœud.demandesRejetées;
    si n = null alors
        // si aucun nœud ne peut traiter la demande, elle est déléguée au cloud.
        d.destination ← cloud;
        envoyerVersCloud(d);
    sinon
        si n est libre alors
            d.destination ← n;
        sinon
            // si le nœud n'est pas libre et que d est plus adéquate que la demande associée à n.
            si distance(n,n.demande) > distance(n,d) alors
                // la destination de la demande associée à n est remise à null pour qu'elle soit
                  retraiter.
                n.demande.destination ← null;
                // puis elle est ajoutée aux demandes rejetées par le nœud pour qu'elle ne puisse pas
                  se reproposer.
                n.demandesRejetées.ajouter(n.demande);
                // la demande associée à n est remplacée par d.
                n.demande ← d;
                d.destination ← n;
            sinon
                // la demande d a été rejetée par le nœud donc elle ne peut plus se reproposer.
                n.demandesRejetées.ajouter(d);
            fin
        fin
    fin
fin
```

Algorithme 2 : Procédure de correspondance

Routine associée au nœud Fog :

Cette routine est exécutée au niveau des nœuds Fog à chaque fois qu'une requête est reçue.

Pseudo-code de la routine :

```
si Requête reçue est de type demande alors
    si demande.destination correspond au nœud lui-même alors
        // la destination du résultat et la source de la demande.
        resultat ← executer(demande) ;
        // puis effectuer le routage du résultat.
        si listeEnfants.contient(resultat.destination) alors
            envoyer(resultat, resultat.destination) ;
        sinon
            envoyer(resultat, NoeudFilsParDefaut) ;
            NoeudFilsParDéfaut ← prochainNoeudFils() ;
        fin
    sinon
        // le nœud en question n'est pas la destination.
        si listeParents.contient(demande.destination) alors
            envoyer(demande, demande.destination) ;
        sinon
            envoyer(demande,NoeudPèreParDéfaut) ;
            NoeudPereParDéfaut ← prochainNoeudParent() ;
        fin
    fin
sinon
    // la requête reçue est par conséquent une requête résultat.
    si listeEnfants.contient(resultat.destination) alors
        envoyer(resultat, resultat.destination) ;
    sinon
        envoyer(resultat, NoeudFilsParDefaut) ;
        NoeudFilsParDéfaut ← prochainNoeudFils() ;
    fin
fin
```

Algorithme 3 : Routine associée aux nœuds passerelles

3.5 Conclusion

Dans ce chapitre, nous avons présenté notre solution en introduisant les entités interagissantes de notre environnement tout en formulant certaines hypothèses sur ce dernier. Nous décrivons aussi le scénario nominal d'exécution et nous détaillons les algorithmes exploités à chaque niveau.

Dans le prochain chapitre, nous passerons à l'étape d'implémentation de la solution sur une plateforme de simulation d'environnement Fog, et nous finirons par une discussion des résultats obtenus.

Chapitre 4

Implémentation et résultats

4.1 Introduction

Dans le chapitre précédent, nous avons présenté toute la théorie mise en place pour la conception de notre solution de planification de ressources relative à la gestion des ressources dans les environnements Fog. Pour cela, nous avons besoin d'outils nous permettant la quantification des performances et la production de résultat, afin de pouvoir effectuer l'évaluation.
Ce chapitre commence par donner un aperçu global sur les différents outils utilisés pour mener à bien la simulation de cette solution, ainsi que les différents éléments ajoutés au simulateur. Enfin nous présentons les résultats en les comparant avec les résultats des différentes politiques gestion classique que sont les politiques : “First fit”, "Best Fit" et "Worst fit".

4.2 Outil de développement

Afin de pouvoir simuler la solution proposée, nous avons opté pour le simulateur “IFogSim”. En effet ce dernier opère suivant le paradigme événementiel, ce qui s'accorde parfaitement avec notre solution qui use du même concept. Développé exclusivement en java, IFogSim nous permet de mesurer l'impact technique résultant de la gestion en nous fournissant des données concernant la consommation d'énergie, les délais d'exécution et l'état du réseau.

4.2.1 Langage JAVA

Java est une technologie initialement développée par la société “Sun Microsystems” en 1995, cette dernière s'est vue rachetée par la suite par la société Oracle en 2009. Java fait office à la fois de langage de programmation orienté objet, mais aussi de machine virtuelle qui permet au langage Java d'être multiplateforme. La technologie java bénéficie d'une grande popularité dû aux avantages considérables proposés par cette dernière parmi lesquels :

- La portabilité : i.e la capacité du même code java à s'exécuter sur plusieurs systèmes d'exploitation.
- Richesses des bibliothèques disponibles
- Gratuité

4.2.2 L'outil IFogSim

IFogSim est un outil de modélisation d'environnements Cloud-Fog-IoT créé par Harshit Gupta (chef de projet) au Georgia Institute of Technology, Atlanta, GA, USA dans l'objectif de simuler l'impact des différentes techniques du management des ressources sur les environnements Fog. IFogSim mesure l'impact d'une politique sur un système par sa latence, l'utilisation de la bande passante, la consommation

d'énergie et l'utilisation des ressources des nœuds. IFogSim a été conçu en tant qu'extension à l'outil CloudSim, qui est un simulateur Open Source d'applications Cloud développé en Java. IFogSim est donc lui aussi Open Source et programmé en Java .

4.2.3 Architecture de l'IFogSim

L'iFogSim est composé de 3 composants principaux, à savoir :
— Les composants physiques.
— Les composants logiques.
— Les composants de gestion.

Composants physiques

Les composants physiques incluent le cloud, les nœuds Fog et les objets, qui sont organisés dans un ordre hiérarchique. Les nœuds Fog de niveau inférieur sont directement connectés aux capteurs et actionneurs associés. Les nœuds Fog agissent comme des centres de traitement de données en offrant de la mémoire et de la puissance de calcul. Chaque nœud Fog est créé avec une puissance de calcul et des attributs de consommation d'énergie spécifique (puissance occupée et inactive), qui reflètent ses capacités et son efficacité énergétique. Les capteurs génèrent des tuples qui peuvent être appelés tâches dans le cloud computing. La création de tuples (tâches) est pilotée par les événements et l'intervalle entre la génération de deux tuples est régi par une distribution déterministe définie lors de la création des capteurs.

Composants logiques

Les modules d'application (AppModules) et les bords d'application (AppEdges) sont les composants logiques d'IFogSim. Dans IFogSim, une application est considérée comme une collection des modules interdépendants, ce qui améliore par conséquent le concept d'application distribuée. La dépendance entre deux modules est définie par les fonctionnalités d'AppEdges qui sont le flux de données logique entre deux modules d'application. Dans iFogSim, chaque module d'application (AppModule) traite un type particulier de tâche (tuple). La transmission de tuples entre deux modules d'application (AppModules) peut être périodique.

Composants de gestion

Les composants de gestion d'IFogSim sont représentés par le « Controller» et le «ModuleMapping». Selon les exigences des AppModules, l'objet ModuleMapping identifie les nœuds Fog associés à chaque AppModules. L'objet Controller lance les AppModules sur leurs nœuds Fog attribués en suivant les informations de placement fournies par l'objet ModuleMapping, et gère périodiquement les ressources des nœuds Fog. Une fois la simulation terminée, l'objet Controller collecte les résultats des coûts, de l'utilisation du réseau et de la consommation d'énergie pendant la période de simulation à partir des nœuds Fog.

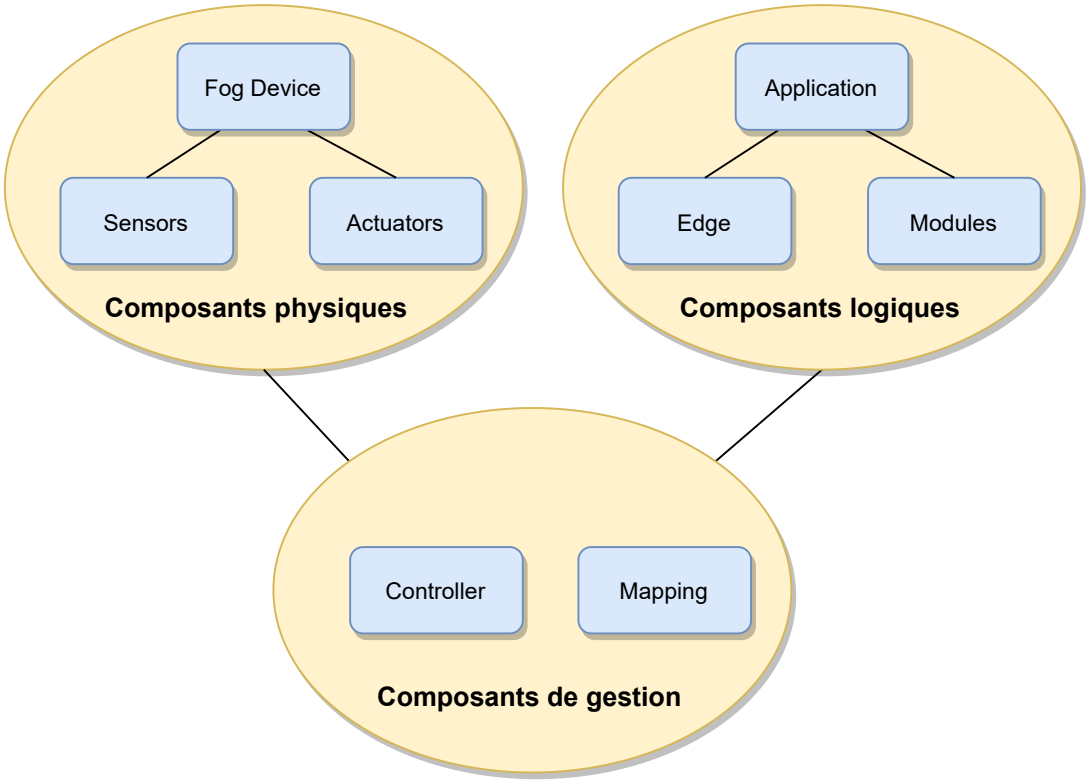

FIGURE 4.1 – Diagramme représentant l'interaction des différents composants de l'IFogSim

Description des classes principales

Les principales classes de l'IFogSim sont les suivantes :

— ***FogDevice :*** Cette classe spécifie les caractéristiques matérielles des nœuds Fog et leurs connexions aux autres nœuds de la topologie. Les principaux attributs de cette classe sont la mémoire accessible, le processeur, la taille de stockage, la bande passante de la liaison montante et la liaison descendante. Les méthodes de cette classe définissent la manière dont les ressources d'un nœud Fog sont planifiées entre les modules qui s'exécutent sur cette application, ainsi que leur déploiement. Par défaut, chaque FogDevice ne peut avoir hiérarchiquement qu'un seul père.

— ***Sensor :*** les instances de la classe Sensor sont des entités qui agissent en tant que capteurs IoT tels que décrits dans l'architecture. La classe contient des attributs représentant les caractéristiques d'un capteur (son nom, le tuple d'émission et un paramètre qui représente les conditions d'émission du Tuple). La classe contient un attribut de référence au dispositif Fog auquel le capteur est connecté et la latence de connexion entre eux.

— ***Actuator :*** - Cette classe modélise un actionneur en définissant l'effet de l'actionnement et ses propriétés de connexion au réseau. La classe définit une méthode pour effectuer une action à l'arrivée d'un tuple à partir d'un module d'application. Un attribut de la classe fait référence à la passerelle à laquelle l'actionneur est connecté et à la latence de cette connexion.

— ***Tuple :*** Les tuples forment l'unité fondamentale de communication entre les entités dans le réseau Fog. Il peut être soit une tâche, soit une donnée. Un tuple est caractérisé par son type, sa source et sa destination. Les attributs de la classe spécifient la puissance de calcul nécessaire à son traitement (mesuré en millions d'instructions (MI)) et la longueur des données encapsulées dans le tuple.

— ***Application :*** Les instances de cette classe représentent les éléments qui composent une application. Pour chaque tuple entrant, une instance AppModule le traite et génère des tuples en sortie, qui sont envoyés aux modules suivants. Le nombre de tuples de sortie par tuple d'entrée est déterminé à l'aide d'un modèle de sélectivité, qui peut être basé sur une sélectivité fractionnaire ou

un modèle éclaté.

- ***AppEdge :*** Une instance AppEdge dénote la dépendance entre une paire de modules d'application. Chaque AppEdge est caractérisé par le type de tuple qu'il transporte, les exigences de traitement et la longueur des données encapsulées dans ces tuples. l'IFogSim prend en charge deux types d' AppEdge (périodique et événementiels). Les tuples dans les AppEdge périodiques sont émis à intervalles réguliers. Le tuple dans un AppEdge est basé sur un événement qui est envoyé lorsque le module source reçoit un tuple précis.
- ***AppLoop :*** est une classe supplémentaire, utilisée pour spécifier les boucles de contrôle de processus qui intéressent l'utilisateur. Dans l'IFogSim, le développeur peut spécifier les boucles de contrôle pour mesurer la latence de bout en bout. Une instance AppLoop est fondamentalement une liste de modules à partir de l'origine de la boucle jusqu'au module où la boucle se termine.

— ***Config :*** Cette classe abstraite a pour but de contenir tous les paramètres qui affectent la simulation par exemple MAX_SIMULATION_TIME qui représente la durée de simulation.

— ***Controller :*** cet objet se charge d'orchestrer la simulation. Concrètement, il sert à initialiser la topologie avant le début de la simulation, ainsi qu'à effectuer les traitements de fin de simulation (notamment afficher et exporter les résultats).

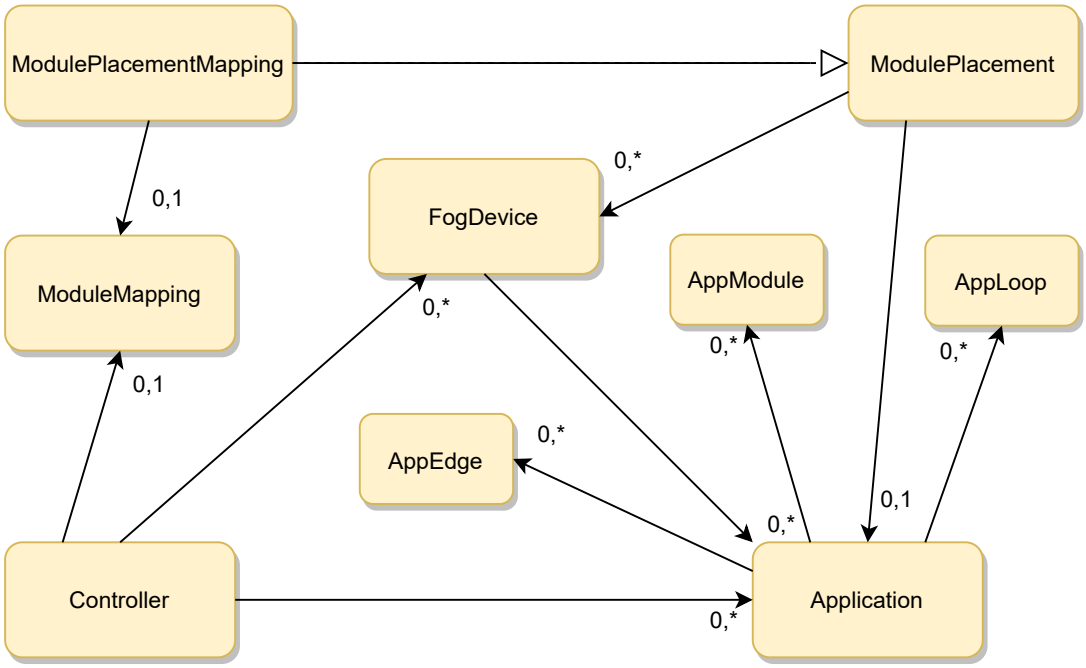

FIGURE 4.2 – Diagramme représentant les relations entre les principales classes de l'IFogSim

4.3 Développement de la conception

Afin de mettre en œuvre la conception proposée, nous allons ajouter de nouvelles classes nécessaires à l'implémentation de la solution, mais aussi la modification de certaines classes principales de l'IFogSim afin de l'adapter à nos besoins.
Les manipulations effectuées pour mettre en oeuvre cette solution sont énumérées ci-dessous :

1. **Création d'une classe ClusterFogDevice :** Cette classe représente un nœud Fog quelconque du cluster. Elle étend la classe prédéfinie FogDevice et comporte en plus les attributs suivant :
   - **parentsIds :** qui est une liste qui contient les identificateurs des nœuds se trouvant au niveau supérieur.
   - **isNorthLinkBusyById :** qui est une liste de booléens où chaque élément indique si la liaison supérieure en question est occupée.

— **northTupleQueues :** qui est une liste de files d'attente de tuple qui associe à chaque liaison supérieure une file de tuples où seront stockés les tuples à envoyer sur cette liaison si cette dernière est occupée.

De plus, la méthode processTupleArrival, qui est la méthode exécutée par le nœud à chaque arrivée d'un tuple, a été redéfinie afin d'implémenter la routine associée à un nœud Fog et qui est décrite dans la conception.

2. **Création d'une classe GWFogDevice :** cette classe représente un nœud Fog passerelle, c.-à-d. le nœud connecté directement aux capteurs et aux actionneurs. Cette classe étend également la classe FogDevice, et ajoute les attributs suivants :
   — **waitingQueue :** qui est une liste de tuples servant à stocker les demandes non encore affectées.
   — **tupleToMatchedDevice :** est une liste contenant les tuples non affectés à un neud.
   — **matchedTupleList :** qui est une liste comportant les tuples affectés à leurs nœuds respectifs.
   — **gwDevices :** est une liste de GWFogDevice, contenant l'ensemble de nœuds passerelles.
   — **isNorthLinkBusyById :** qui représente la même chose que dans la classe.
   — **northTupleQueues :** elle représente également la même chose que celle mentionnée dans la classe ClusterFogDevice.
   — **clusterFogDevicesIds :** qui représente la liste de tous les nœuds Fog du cluster.
3. **Création d'une classe MatchedTuple :** cette classe décrit les tuples une fois affectés à leurs nœuds respectifs. Elle représente un enrichissement de la classe Tuple par les attributs suivants :
   — **destinationFogDeviceId :** qui contient l'identificateur du nœud de destination.
   — **destModuleMips :** qui décrit les exigences de traitement du module de destination.

La même procédure à été effectueé pour implémenter les politiques concurrentes.

4.4 Résultats et évaluation des performances :

Afin de pouvoir juger des performances ainsi que l'ampleur des gains apportés par notre modèle de planification de ressources, nous évaluons les performances de cette solution en mesurant le délai d'exécution des applications, et le délai d'exécution des tuples. Puis nous comparons les résultats avec des modèles classiques implémentant les stratégies “FirstFit”,"BestFit","WorstFit".

Les nœuds *Fog* ainsi que les demandes générées sont créés avec une certaine hétérogénéité permettant de mieux étudier la capacité de notre algorithme à faire correspondre une variété de demandes à des nœuds avec de différentes capacités matérielles.

Les résultats sont représentés par les graphes suivants (les résultats sont normalisés par rapport à la valeur maximale, donc sur une échelle de 0 à 1) :

Scalabilité verticale

Nous mesurons dans cette partie les performances du modèle par rapport à la variation du nombre de niveaux de la matrice *Fog* précédemment définie.

Nous fixons dans cette expérience le nombre de nœudspar niveau à 5 nœuds.

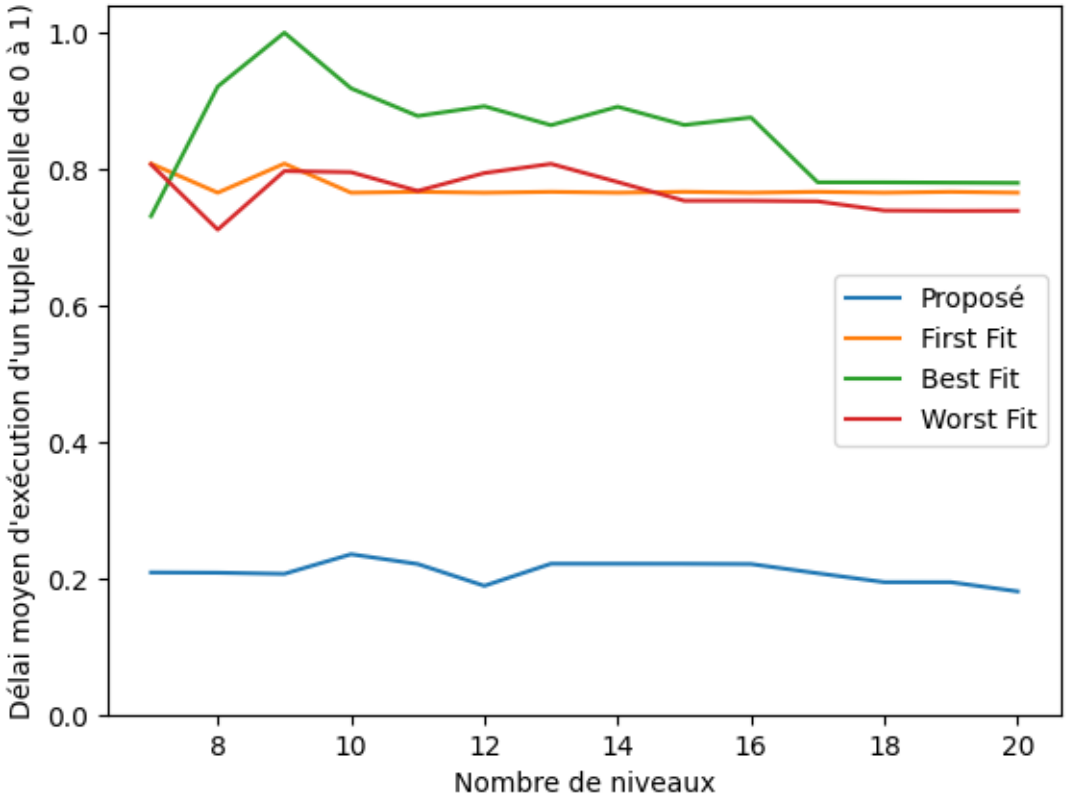

FIGURE 4.3 – Délai moyen d'exécution d'un tuple en fonction du nombre de niveaux

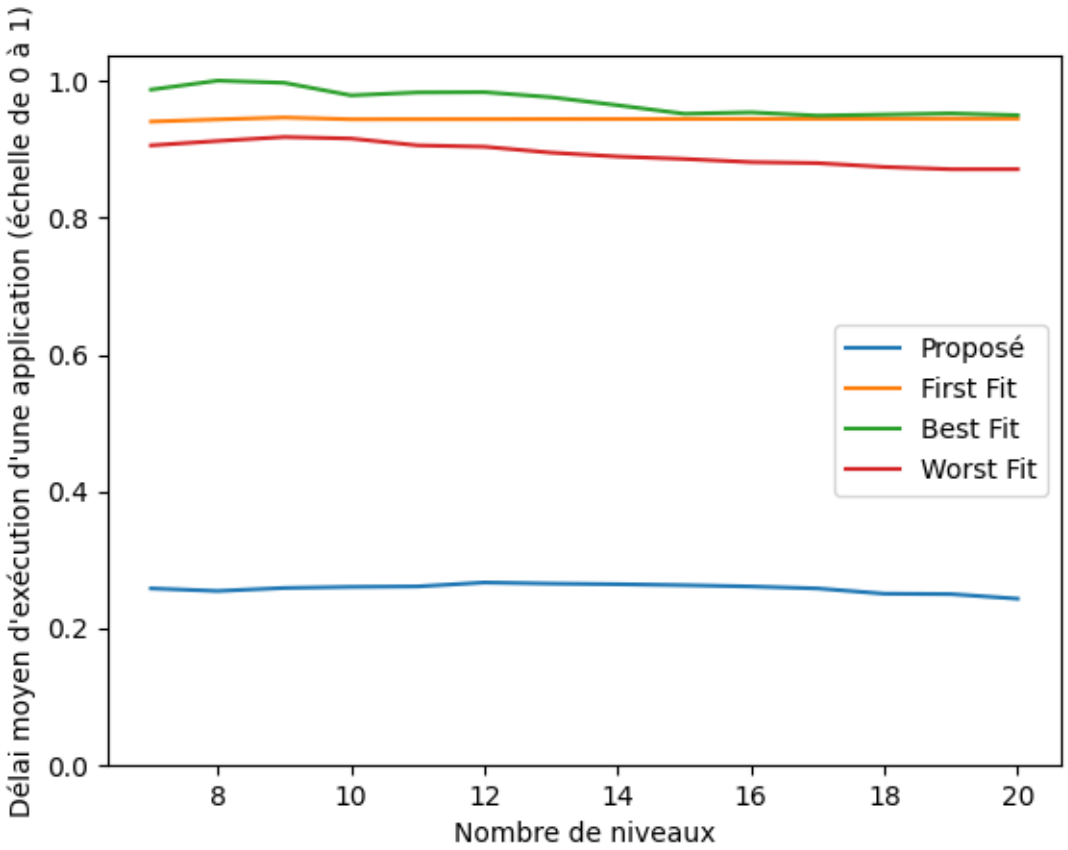

FIGURE 4.4 – Délai moyen d'exécution d'une application en fonction du nombre de niveaux

Les figures 4.3 et 4.4 montrent, respectivement, la variation du délai moyen de l'exécution d'un tuple (représentant une demande de service) et le délai d'exécution d'une application complète selon la nombre de niveaux de l'architecture.

Nous constatons une nette réduction du délai de traitement des tuples d'environ 75%, en comparaison aux autres algorithmes de la simulation. Cette réduction se stabilise au-delà du seuil de 18 niveaux, où le nombre de niveaux n'influence plus cette métrique. De même pour le délai d'exécution d'une application, où nous pouvons constater l'impact de la réduction du temps de réponse des tuples individuels.

Scalabilité horizontale

Nous mesurons dans cette partie les performances du modèle par rapport à la variation du nombre de nœuds par niveau de la matrice *Fog* précédemment définie.

Le nombre de nœuds par niveau influence le nombre de tuples générés par les objets IoT car chaque nœud passerelle est relié à un ensemble d'objets IoT. Ainsi l'augmentation du nombre de nœuds passerelle augmente aussi le nombre de demandes générées.

Nous fixons le nombre de niveaux à 10 niveaux.

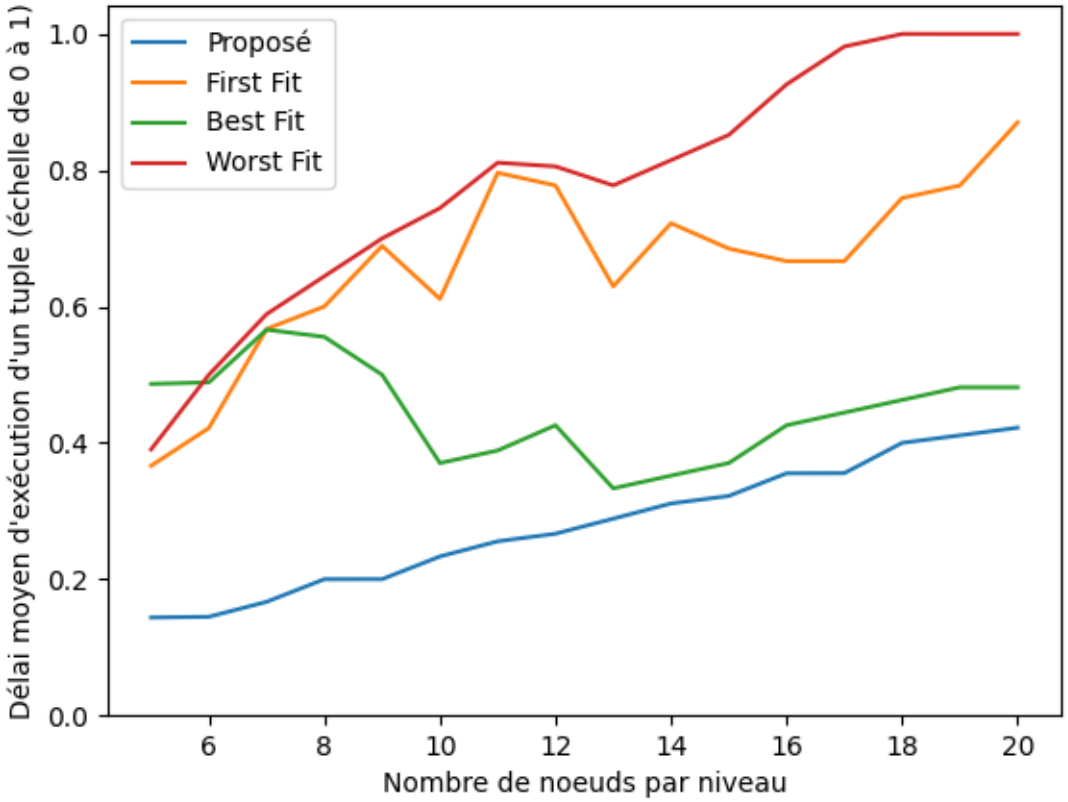

FIGURE 4.5 – Délai moyen d'exécution d'un tuple en fonction du nombre de nœuds par niveaux

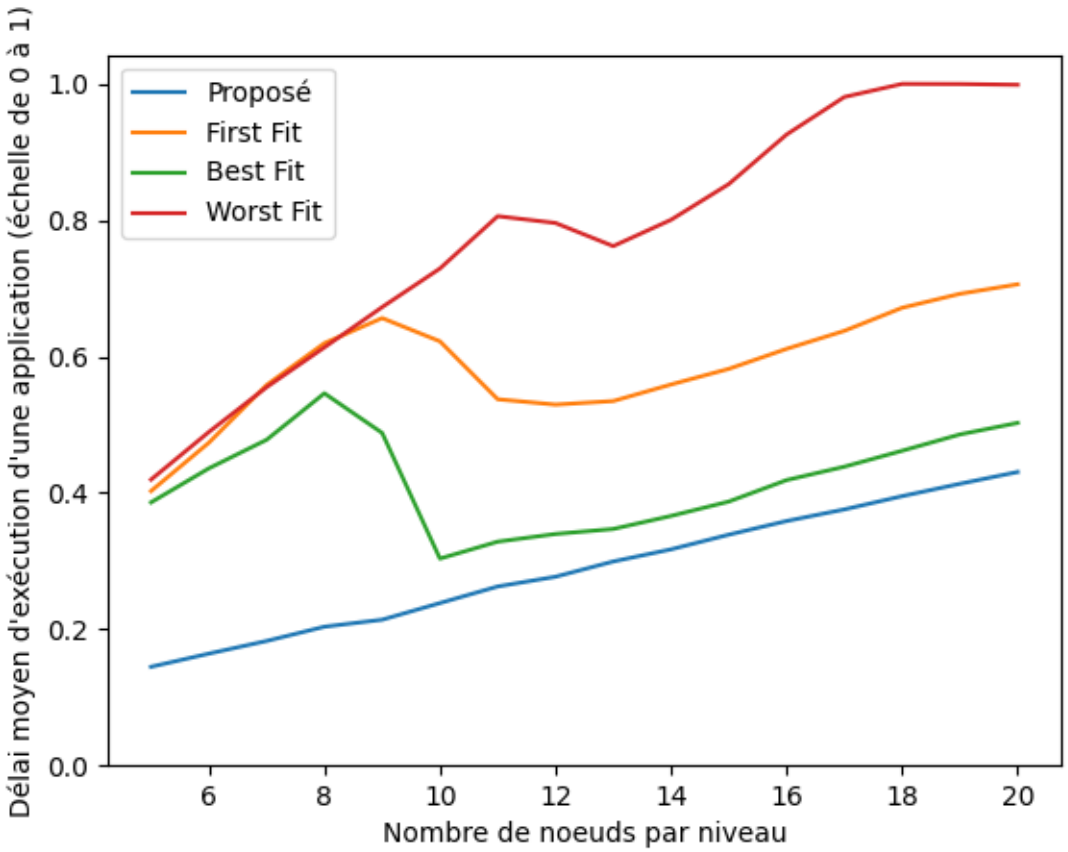

FIGURE 4.6 – Délai moyen d'exécution d'une application en fonction du nombre de nœuds par niveau

Les figures 4.5 et 4.6 représentent, respectivement, la variation du délai de l'exécution d'un tuple et le délai d'exécution d'une application selon le nombre de nœuds par niveau.

La figure 4.5 montre un délai d'exécution croissant linéairement de la part des stratégies classiques, contrairement à la méthode proposée qui semble suivre une tendance logarithmique. Nous concluons par conséquent que pour les stratégies classiques, l'augmentation du nombre de nœuds par niveau influe significativement sur le délai d'exécution d'une application, par opposition à la stratégie proposée ou la corrélation semble moins significative.

Nous remarquons que pour la stratégie "Best Fit", la croissance de la durée d'exécution d'un tuple est linéairement croissante en fonction du nombre de nœuds par niveau. Pour la stratégie "First Fit", la durée d'exécution augmente jusqu'à se stabiliser quand le nombre de nœuds dépasse 19 nœuds par niveau. Ainsi que pour la stratégie "Worst Fit", nous constatons une augmentation suivie d'une phase de diminution à partir de 17 nœuds par niveaux. Enfin la stratégie proposée semble suivre une cadence doucement haussière en ayant de meilleurs résultats, ainsi qu'une meilleure stabilité du temps d'exécution.

Exploitation des résultats des expériences sur la scalabilité pour déduire le schémas de clusterisation

Nous observons, à partir de l'union des deux expériences précédentes, une baisse des performances dès que le nombre de nœuds passerelles dépasse le nombre de niveaux du cluster Fog.

Afin que l'effet de la clusterisation soit bénéfique, ce dernier se doit de respecter la contrainte suivante :
Soit N et M le nombre de nœuds passerelles et le nombre de niveaux (y compris les nœuds passerelles) respectivement.

Soit n_i le nombre de nœuds passerelles pour le cluster numéro i des p clusters.

$$\forall i \leq p,\ n_i \leq \mathrm{M} - 1$$

Étude selon le débit des demandes

Dans ce qui suit, nous étudions les performances de notre modèle selon le délai entre les envois de chaque demande. Naturellement, plus le délai entre les demandes est petit, plus il y a de charge et de congestion dans le réseau.

Nous fixons dans cette expérience le nombre de niveaux à 7 niveux et le nombre de nœuds par niveau à 5 nœuds ($5 \leq 7 - 1$).

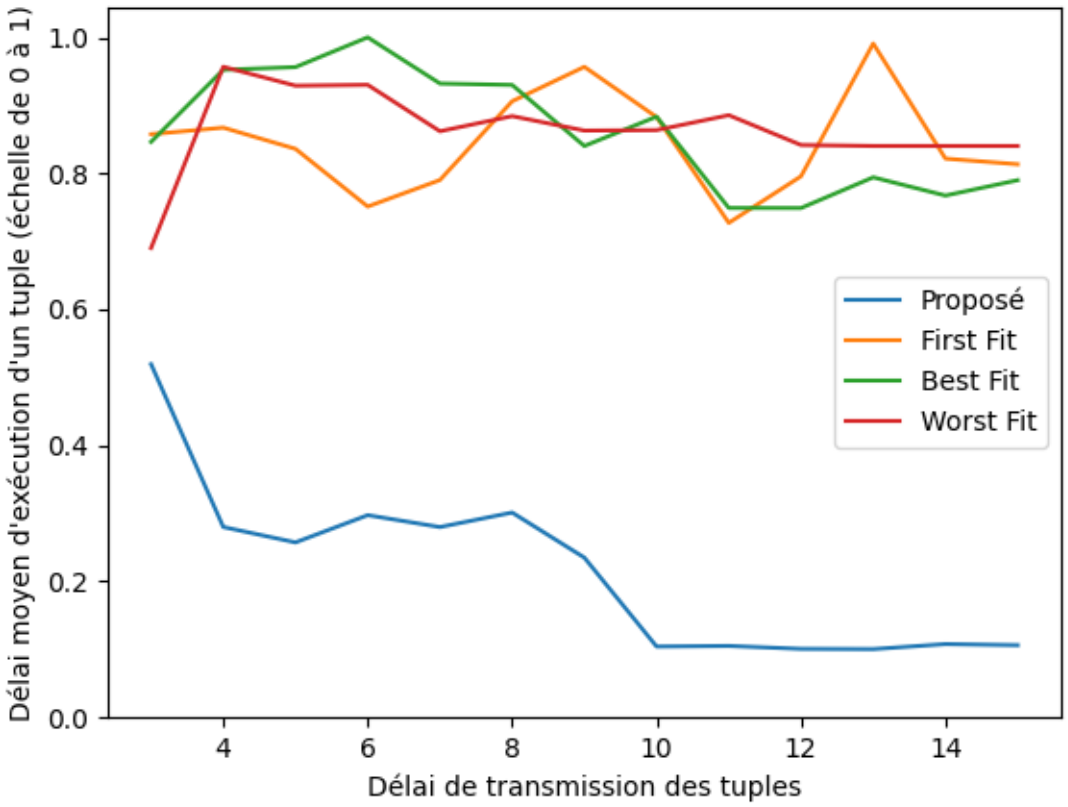

FIGURE 4.7 – Délai moyen d'exécution d'un tuple en fonction du taux de transmission

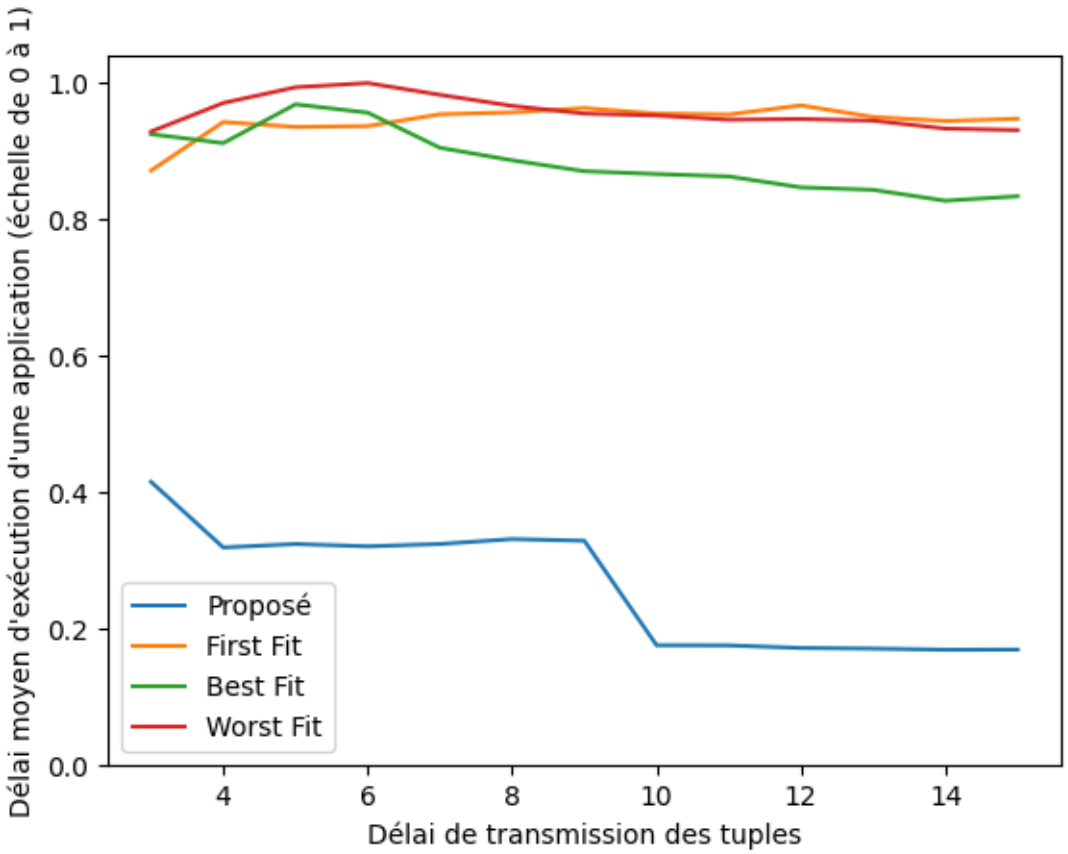

FIGURE 4.8 – Délai moyen d'exécution d'une application en fonction du taux de transmission

Les figures 4.7 et 4.8 représentent, respectivement, la variation du délai d'exécution d'un tuple et le délai d'exécution d'une application selon le délai de transmission entres les tuples consécutifs. Nous constatons que la durée d'exécution d'une application dans les stratégies classiques est stable et proche de la valeur maximale, ce qui signifie que la diminution de la charge de travail, n'influe pas sur la durée d'exécution. En revanche, la stratégie proposée démontre des performances meilleures en termes de délai ainsi qu'une réaction à la variation du délai de transmission à partir de 9 unités de temps de délai.

4.5 Conclusion

Nous avons réussi à montrer l'efficacité de notre technique dans la réduction du temps d'exécution des demandes de services, témoignant ainsi d'une meilleure gestion et répartition de ressources de l'ensemble des nœuds du cluster.

Chapitre 5

Conclusion générale

Dans ce mémoire nous avons étudié le paradigme du Fog Computing qui évolue rapidement pour atténuer les problèmes de latence, de bande passante et de qualité de service (QoS) des applications basées sur le cloud. Nous avons présenté les efforts de recherche axés sur l'optimisation de l'exploitation des ressources que propose cette architecture.

A notre tour, nous avons proposé un algorithme décentralisé pour la distribution équitable des demandes de services par lot sur l'ensemble de nœuds Fog organisés en clusters. Notre solution favorise la réactivité et la réduction du temps de réponse aux demandes, qui est un facteur critique pour les applications à temps réel qui expriment de grandes exigences de qualité de service. De plus, cette solution garantit la distribution équitable de charges de travail entre les nœuds d'un même cluster, ce qui permet d'éviter la saturation du réseau et la création de points critiques.

La solution présentée traite seulement le problème d'allocation de ressources, et peut être enrichie en explorant d'autres aspects de l'architecture Fog. Nous pouvons énumérer les perspectives suivantes permettant notamment d'éliminer les hypothèses formulées dans le chapitre de conception :

— L'intégration d'un mécanisme de découverte du réseau permettant la construction d'une topologie logique de ce dernier. Ceci nous permettra de considérer le coût de la liaison réseau dans la fonction de distance entre les demandes et les nœuds proposée dans le chapitre de conception.
— Permettre l'ajout et le retrait dynamique de nœuds du cluster à l'aide d'une solution d'abonnement aux nœuds passerelles.
— Intégrer la notion de priorité entre les différentes demandes émanant des objets IoT.

Chapitre 6

Annexe

Notion de stabilité : On dit d'une solution de matching S qu'elle est stable ssi

$$\neg\exists(m;w),(m';w') \in S,\ m\ préfère\ w' \text{ à } w\ et\ w'\ préfère\ m \text{ à } m'$$

Bibliographie

[1] "Quelle est la différence entre les conteneurs et les machines virtuelles?" Alibaba Cloud, accessed at 18/01/2021. [Online]. Available : https://www.alibabacloud.com/fr/knowledge/difference-between-container-and-virtual-machine

[2] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, "Container migration in the fog : A performance evaluation," *Sensors*, vol. 19, p. 1488, 03 2019.

[3] K. K. Andrey Mirkin, Alexey Kuznetsov, "Containers checkpointing and live migration," in *Proceedings of the Linux Symposium, vol. 2*, Ottawa, Ontario. Canada, 2008, pp. 85–90.

[4] K. K. Patel, S. M. Patel *et al.*, "Internet of things-iot : definition, characteristics, architecture, enabling technologies, application & future challenges," *International journal of engineering science and computing*, vol. 6, no. 5, 2016.

[5] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of things (iot) : A literature review," *Journal of Computer and Communications*, vol. 3, pp. 164–173, 04 2015.

[6] K. Ashton *et al.*, "That 'internet of things' thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[7] A. Čolaković and M. Hadžialić, "Internet of things (iot) : A review of enabling technologies, challenges, and open research issues," *Computer Networks*, vol. 144, pp. 17–39, 2018. [Online]. Available : https://www.sciencedirect.com/science/article/pii/S1389128618305243

[8] 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*, 2012, pp. 13–16.

[9] C. B. Robert Swanson *et al.*, "Openfog reference architecture for fog computing," 2017.

[10] M. De Donno, K. Tange, and N. Dragoni, "Foundations and evolution of modern computing paradigms : Cloud, iot, edge, and fog," *Ieee Access*, vol. 7, pp. 150 936–150 948, 2019.

[11] R. Boutaba, Q. Zhang, and M. F. Zhani, *Virtual Machine Migration in Cloud Computing Environments : Benefits, Challenges, and Approaches*, 01 2013, pp. 383–408.

[12] "Conteneurs et machines virtuelles," Red Hat, accessed at 18/01/2021. [Online]. Available : https://www.redhat.com/fr/topics/containers/containers-vs-vms

[13] J. Gerend, "Conteneurs ou machines virtuelles," Microsoft, Oct. 21, 2019, accessed at 18/01/2021. [Online]. Available : https://docs.microsoft.com/fr-fr/virtualization/windowscontainers/about/containers-vs-vm

[14] N. Chandrakala and D. B. Rao, "Migration of virtual machine to improve the security in cloud computing," *International Journal of Electrical and Computer Engineering*, vol. 8, pp. 210–219, 02 2018.

[15] "Migration de machines virtuelles," Wikipédia français, accessed at 21/01/2021. [Online]. Available : https://fr.wikipedia.org/wiki/Migration_de_machines_virtuelles#Migration_de_m%C3%A9moire_par_post-copie_hybride

[16] Z. Rejiba, X. Masip, and E. Marin-Tordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Computing Surveys*, vol. 52, pp. 1–33, 09 2019.

[17] R. B. Redowan Mahmud, Kotagiri Ramamohanarao, "Application management in fog computing environments : A taxonomy, review and future directions," *ACM Computing Surveys*, vol. 53, pp. 1–8, May 2020.

[18] A. A. R. Mostafa Ghobaei-Arani, A. Souri, "Resource management approaches in fog computing : a comprehensive review," *Journal of Grid Computing*, 2019.

[19] N. K. R. Ashish Virendra Chandak, "A review of load balancing in fog computing," *2019 International Conference on Information Technology*, 2019.

[20] B. V. Cheol-Ho Hong, "Resource management in fog/edge computing : A survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys*, vol. 52, no. 5, sep 2019.

[21] C.-T. C. C. K. T. Jing V. Wang, Kai-Yin Fok, "A stable matching-based virtual machine allocation mechanism for cloud data centers," *2016 IEEE World Congress on Services*, July 2016.

[22] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, Jan 1962.

Conception et implémentation d'une technique d'allocation de ressources dans un environnement *Fog Computing*

RÉSUMÉ :

Le fog computing, l'informatique géodistribuée, l'informatique en brouillard, ou encore l'infonébulisation, consiste à exploiter des applications et des infrastructures de traitement et de stockage de proximité, servant d'intermédiaire entre des objets connectés et une architecture informatique en nuage classique. Le but est d'optimiser les communications entre un grand nombre d'objets connectés et des services de traitement distants, en tenant compte d'une part des volumes de données considérables engendrés par ce type d'architecture (mégadonnées) et d'autre part de la variabilité de la latence dans un réseau distribué, tout en donnant un meilleur contrôle sur les données transmises.

MOTS-CLEFS : XXXX - YYYYY - ZZZZ .

ABSTRACT :

Fog computing, also called Edge Computing, is intended for distributed computing where numerous "peripheral" devices connect to a cloud. (The word "fog" suggests a cloud's periphery or edge). Many of these devices will generate voluminous raw data (e.g., from sensors), and rather than forward all this data to cloud-based servers to be processed, the idea behind fog computing is to do as much processing as possible using computing units co-located with the data-generating devices, so that processed rather than raw data is forwarded, and bandwidth requirements are reduced. An additional benefit is that the processed data is most likely to be needed by the same devices that generated the data, so that by processing locally rather than remotely, the latency between input and response is minimized. This idea is not entirely new : in non-cloud-computing scenarios, special-purpose hardware (e.g., signal-processing chips performing Fast Fourier Transforms) has long been used to reduce latency and reduce the burden on a CPU.

KEYWORDS : XXXX - YYYYY - ZZZZ .