Conception d'un système de supervision programmable et reconfigurable pour une infrastructure informatique et réseau répartie

Mohamed Abderrahim

▶ **To cite this version:**

Mohamed Abderrahim. Conception d'un système de supervision programmable et reconfigurable pour une infrastructure informatique et réseau répartie. Autre [cs.OH]. Ecole nationale supérieure Mines-Télécom Atlantique, 2018. Français. NNT : 2018IMTA0119 . tel-02049129

HAL Id: tel-02049129

https://theses.hal.science/tel-02049129v1

Submitted on 26 Feb 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE
COMUE UNIVERSITE BRETAGNE LOIRE

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

Par

« Mohamed ABDERRAHIM »

« Conception d'un système de supervision programmable et reconfigurable pour une infrastructure informatique et réseau répartie »

Thèse présentée et soutenue à « Nantes », le « 19/12/2018 »
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)
Thèse N° : 2018IMTA0119

Rapporteurs avant soutenance :

Frederic Desprez	Directeur de recherche, Inria
Christian Perez	Directeur de recherche, Inria

Composition du Jury :

Président :	Frederic Desprez	Directeur de recherche, Inria
Examinateurs :	Pierrick Seite	Ingénieur de recherche (docteur), Orange Labs
	Christian Perez	Directeur de recherche, Inria
Dir. de thèse :	Adrien Lebre	Professeur, IMT-Atlantique
Co-encadrants :	Karine Guillouard	Ingénieur de recherche (docteur), Orange Labs
	Jérôme François	Chargé de recherche, Inria

Remerciements

Mener cette thèse a été une expérience cruciale pour moi et cela n'aurait pas été possible sans le soutien et les conseils que j'ai reçus de nombreuses personnes.

Tout d'abord, je voudrais exprimer ma gratitude à mon directeur de thèse Adrien Lebre pour ses encouragements et ses recommandations inestimables. Je souhaite exprimer ma grande reconnaissance également à mes encadrants Karine Guillouard, Meryem Ouzzif et Jérôme François pour leur disponibilité et leurs conseils indispensables.

Je tiens à remercier les membres de mon jury de thèse, Frédéric Desprez, Christian Perez et Pierrick Seite qui m'ont fait l'honneur d'accepter de juger mes travaux.

Un grand merci à tous les membres de l'équipe AMI d'Orange Labs pour leur encouragement et leur aide. Je remercie particulièrement le responsable de cette équipe Florent Le Lain pour sa confiance.

Merci à tous les membres de l'équipe STACK de l'Inria pour leur accueil lors de mes déplacements à Nantes et pour les différentes activités sportives que nous avons exercées ensemble.

Merci à Charles Prud'homme et à Xavier Lorca de l'équipe TASC de l'IMT-Atlantique pour leur aide dans la modélisation du placement mutualisé.

Un grand merci à mon cher oncle Chettaoui pour avoir beaucoup voulu être à mes côtés lors de la soutenance de ma thèse, à mon cher ami Anas de s'être déplacé depuis Paris pour y assister et à tous mes amis pour leur soutien tout au long de ces trois dernières années.

Un immense merci à mes parents, ma sœur et mon frère qui m'ont donné le courage et la persévérance d'aller jusqu'au bout de ce long voyage. Je suis sensible à leur patience et à leur amour inconditionnel. Cette réalisation n'aurait pas été possible sans eux.

Table des matières

Introduction

Les applications Cloud jouent un rôle très important dans notre vie quotidienne. Notre usage de plusieurs d'entre elles comme les moteurs de recherche, les réseaux sociaux ou les plateformes de partage de contenus multimédia n'a pas cessé de se développer. En effet, le Cloud représente un succès incontestable dont en témoigne le chiffre d'affaires de son marché qui a atteint 180 milliards de dollars en 2017 [Synergy 2018]. Son principe consiste à offrir différents types de ressources (par exemple, calcul, stockage et réseau) en tant que services. Il dispense, ainsi, ses utilisateurs du maintien d'une infrastructure physique et de toutes les contraintes associées (par exemple, achat et renouvellement du matériel, alimentation, mise à jour du logiciel, sauvegarde). Les utilisateurs n'ont qu'à payer selon le modèle économique "payer à l'usage" (Pay as you go) [Rolffs et al. 2015]. Ainsi, le calcul, stockage et réseau sont payés forfaitairement ou en fonction de la consommation tout comme l'électricité ou l'eau.

Afin de réduire le coût des services Cloud, les opérateurs ont tendance à mettre en place une économie d'échelle en s'appuyant sur des infrastructures centralisées. Par conséquent, les requêtes des utilisateurs sont toutes envoyées à des centres de données centralisés et potentiellement éloignés des utilisateurs. Cette centralisation entrave la réalisation des performances des réseaux de 5ème génération (5G) qui ciblent une latence inférieure à 5 ms et une bande passante qui peut aller jusqu'à 10 Gb/s [5G PPP 2015]. De telles performances sont indispensables pour héberger les applications de nouvelle génération (par exemple, l'automatisation de l'industrie, les systèmes de transport ou les réseaux électriques intelligents) qui nécessitent une faible latence et un haut débit.

Pour faire face à cette limite, des experts du monde académique et industriel recommandent de rompre avec le modèle des infrastructures purement centralisées. Ils proposent d'étendre le Cloud à des infrastructures massivement distribuées qui peuvent être déployées jusqu'à la périphérie du réseau. Ce nouveau type d'infrastructures, référencé à ce jour sous le paradigme dit de Edge Computing, constitue une composante fondamentale de la 5G [Khodashenas et al. 2017]. Toutefois, l'essor de ce nouveau paradigme nécessite la mise en œuvre de gestionnaires capables d'opérer et de proposer à des utilisateurs externes les ressources géo-distribuées constituant l'infrastructure Edge. Bien que des solutions opérationnelles exploitant les bénéfices de la géo-distribution apparaissent (Akamai+Cloudlet [1] ou Amazon Lambda@Edge [2]), elles restent limitées à des applications spécifiques (par exemple, équilibrage de charge réseau, pare-feu, etc.) et sont encore très loin d'offrir le même niveau de fonctionnalités que les plateformes qui ont fait le succès des infrastructures Cloud. Ceci s'explique en partie par le fait que l'extension des solutions de gestion destinées aux infrastructures Cloud telles qu'OpenStack [3] ou OpenMANO [4] n'est pas évidente, la plupart d'entre elles ayant été conçues pour des architectures de ressources centralisées [Lebre et al. 2017].

Proposer un canevas logiciel pour l'ensemble des fonctionnalités que doit assurer un gestionnaire d'infrastructures Edge est un travail considérable qui demande des expertises

[1] http ://cloudlets.akamai.com (visité le 01/10/2018)
[2] https ://aws.amazon.com/lambda/edge/ (visité le 01/10/2018)
[3] https ://www.openstack.org/ (visité le 01/10/2018)
[4] https ://osm.etsi.org/ (visité le 01/10/2018)

dans différents domaines. C'est notamment un des objectifs du laboratoire joint *<I/O> lab* entre la société Orange et l'institut Inria dans lequel s'inscrit ce doctorat en CIFRE. Dans cette thèse, nous nous sommes intéressés d'une manière plus spécifique à la supervision qui est une fonction clé d'un gestionnaire de ressources. Celle-ci est utilisée, par exemple, pour garantir la qualité de service, détecter les pannes, provisionner les ressources ou assurer la sécurité. Elle est d'autant plus importante dans le cas d'infrastructures Edge où les ressources sont plus hétérogènes, réparties et dynamiques. Néanmoins, maintenir une fonction de supervision peut être une tâche très complexe et coûteuse. D'une part, il est nécessaire de l'adapter aux différents types de ressources observées indépendamment de leur technologie, de remonter l'état des ressources à une fréquence élevée, voire en temps réel, et d'assurer des propriétés de robustesse comme le passage à l'échelle et la résilience aux pannes. D'autre part, l'empreinte de cette fonction sur l'infrastructure risque d'être non négligeable. À titre d'exemple, la supervision de l'infrastructure de Twitter génère 2.8 milliards de mesures par minute [Asta 2016]. Cette empreinte ne se limite pas au calcul et concerne également le réseau étant donné l'aspect massivement distribué de l'infrastructure Edge. Pour ces raisons, de nombreux utilisateurs optent pour des solutions qui offrent la supervision en tant que service utilitaire [Romano et al. 2011] comme CloudWatch [5], Datadog [6] ou encore New Relic [7]. Cependant, aucun de ces services n'a été conçu en prenant en considération les spécificités de l'infrastructure Edge comme nous allons l'aborder dans ce manuscrit.

La contribution de cette thèse est la proposition d'un canevas logiciel pour la mise en place d'un service de supervision holistique. Ce service doit pouvoir être utilisé par tous les tenants de l'infrastructure Edge, (*c.-à-d.*, l'opérateur, les fournisseurs d'infrastructure, les fournisseurs de services et les utilisateurs de services) afin d'observer l'état de leurs ressources (par exemple, machine physique, passerelle résidentielle, application SaaS, capteur de température). Le canevas proposé permet de déterminer l'architecture de déploiement des fonctions de base de supervision : observation, traitement et exposition des mesures. Il les structure selon une architecture décentralisée qui permet de satisfaire cinq propriétés : la compatibilité avec les différentes ressources hôtes, le passage à l'échelle, la tolérance aux pannes des serveurs, la tolérance aux pannes réseau et la proximité aux utilisateurs. Cette architecture est déterminée à partir d'un calcul de placement qui tient compte à la fois des contraintes fonctionnelles et des qualités de service de l'ensemble des utilisateurs. Étant donné la grande empreinte de calcul et réseau qu'une supervision holistique risque d'introduire sur l'infrastructure, nous avons proposé une approche pour la réduire lors du calcul du placement. Cette approche consiste à mutualiser les flux et les fonctions identiques entre des utilisateurs différents qui requièrent l'observation d'une même ressource. Cette mutualisation est favorisée par l'aspect multi-tenant de l'infrastructure Edge qui permet le partage de ressources entre des utilisateurs différents (par exemple, une passerelle résidentielle à la périphérie du réseau peut être partagée entre son propriétaire et l'opérateur). Nous avons choisi de modéliser le calcul du placement

[5] https ://aws.amazon.com/cloudwatch/ (visité le 01/10/2018)
[6] https ://www.datadoghq.com/ (visité le 01/10/2018)
[7] https ://www.newrelic.com/ (visité le 01/10/2018)

par un problème de satisfaction de contraintes qui reçoit en entrée une description de l'infrastructure et des besoins de supervision des utilisateurs. L'hétérogénéité de l'infrastructure Edge ainsi que la diversité des besoins de supervision de ses utilisateurs constituent des challenges à notre approche. Pour une première mise en œuvre, nous nous sommes appuyés sur deux langages de description que nous avons intégrés à notre proposition.

Cette thèse a donné lieu à la publication et la présentation de l'analyse des solutions de supervision existantes à la conférence IEEE FiCloud en 2017 [Abderrahim et al. 2017]. Le modèle du placement mutualisé et son évaluation ont été acceptés pour publication et présentation à la conférence Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Pavie, 2019. Par ailleurs, ces travaux ont été présentés dans des événements nationaux tels que la journée des doctorants organisée par l'école doctorale ED STIM en 2017 et la journée des doctorants organisée par Orange en 2017. Un premier prototype, notamment un calculateur de placement est disponible en ligne à l'adresse (`https://github.com/edgeMonitoring/PlacementCalculator`).

Ce manuscrit de thèse est organisé en trois parties. La première partie est articulée autour de deux chapitres qui présentent le contexte et l'état de l'art. Dans le chapitre 1, nous introduisons les principes du Edge Computing qui est une réponse à l'incapacité du Cloud Computing à satisfaire les besoins des applications de nouvelle génération. Nous décrivons l'infrastructure sur laquelle s'appuie ce nouveau paradigme et nous identifions les caractéristiques principales qui complexifient la gestion de cette infrastructure. Nous clôturons ce chapitre en présentant le service de supervision holistique que nous ciblons. Dans le chapitre 2, nous dressons l'état de l'art. Nous spécifions les propriétés fonctionnelles et non fonctionnelles que les solutions existantes doivent satisfaire pour mettre en œuvre la supervision de l'infrastructure Edge. Ensuite, nous évaluons qualitativement les différentes solutions par rapport à ces propriétés. Enfin, nous clôturons le chapitre par une synthèse.

Dans la deuxième partie de ce manuscrit, nous détaillons notre contribution qui consiste en un canevas logiciel pour la mise en place d'un service de supervision holistique de l'infrastructure Edge. Dans le chapitre 3, nous présentons les différents modules qui constituent ce canevas. Parmi eux, un module est chargé du placement des fonctions de supervision sur l'infrastructure Edge. Il mutualise les fonctions et les flux identiques entre les différents tenants en vérifiant la satisfaction de leurs contraintes fonctionnelles et de qualité de service. Dans le chapitre 4, nous montrons que le calcul réalisé par ce module se ramène à la résolution d'un problème de satisfaction de contraintes. Nous détaillons son modèle et évaluons sa capacité à réduire l'empreinte du service de supervision sur les ressources de l'infrastructure tout en considérant le coût qu'il introduit en termes de temps de calcul.

Enfin, dans la troisième partie de ce manuscrit, nous concluons cette thèse et proposons plusieurs perspectives pour poursuivre les travaux entrepris. Dans le chapitre 5, nous rappelons les principaux points abordés dans les différents chapitres. Dans le chapitre 6, nous identifions cinq axes de recherche qui concernent l'évaluation des langages de description définis, le calcul de mutualisation, la mise en œuvre de l'architecture de déploiement, l'application de la mutualisation dans le contexte de l'Internet des Objets

et la coopération entre différents gestionnaires de services.

Contexte & État de l'art

1

Contexte

Sommaire

Le paradigme du Cloud Computing est un grand succès en raison des avantages techniques et économiques qu'il offre à ses utilisateurs. Cependant, ce paradigme n'est pas adapté pour héberger les applications de nouvelle génération qui ont des exigences fortes en termes de latence et en bande passante. Pour cela, un nouveau paradigme est apparu. Il s'agit du Edge Computing. Il cherche à étendre le Cloud Computing pour garder ses avantages en dépassant ses limites.

Dans ce chapitre, nous commençons par présenter le Cloud Computing. Ensuite, nous justifions son incapacité à satisfaire les besoins des applications de nouvelle génération

qui ont des exigences strictes en latence et de bande passante. Puis, nous présentons le Edge Computing qui cherche à dépasser les limites du Cloud en s'appuyant sur des ressources déployées à proximité des utilisateurs. Enfin, nous identifions les défis à relever pour réussir la mise en place du nouveau paradigme.

1.1 Informatique en nuage ou Cloud Computing

L'informatique en nuage ou le Cloud Computing permet d'exposer le calcul, le stockage et le réseau en tant que services accessibles à distance. Ainsi, il dispense ses utilisateurs du maintien d'une infrastructure physique et des contraintes associées : mise à jour du logiciel, sauvegarde, achat et renouvellement du matériel, alimentation ... En plus, le Cloud maintient une qualité pour les services offerts selon un contrat (SLA) établi avec les utilisateurs. Ce contrat spécifie essentiellement les critères de disponibilité des services. Le fournisseur du Cloud se charge d'assurer toutes les propriétés nécessaires (par exemple, la sécurité, le passage à l'échelle, la tolérance aux pannes matérielles et dysfonctionnements logiciels) pour maintenir la disponibilité spécifiée indépendamment des circonstances externes. En plus de ces avantages techniques, le Cloud offre un avantage économique aux utilisateurs. En effet, pour bénéficier des services Cloud, les utilisateurs payent uniquement en fonction de leur utilisation selon le modèle économique "payer à l'usage" (Pay as you go) [Rolffs et al. 2015]. Ce modèle leur donne une grande flexibilité de consommation qui les encourage à utiliser le Cloud.

1.1.1 Modèles de services du Cloud

Les services Cloud peuvent être classés en trois modèles historiques selon la nature des ressources offertes :

Application en tant que service (Software as a Service (SaaS))

Dans ce modèle, la ressource exposée en tant que service est une application. Il peut s'agir d'une application de messagerie (par exemple, Gmail [1]), une application bureautique (par exemple, Microsoft Office 365 [2]) ou une application de stockage de données (par exemple, Dropbox [3]). Ainsi, les applications n'ont plus besoin d'être installées en local. Pour y accéder, il suffit d'utiliser un client web.

Plate-forme en tant que service (Platform as a Service (PaaS))

Dans ce modèle, la ressource exposée est une plate-forme qui permet de construire, exécuter ou gérer des applications. Ainsi, ce service dispense l'utilisateur de la mise en place des environnements matériels et logiciels nécessaires pour déployer ses applications.

[1] https ://www.google.com/gmail/ (visité le 01/10/2018)
[2] https ://www.office.com/ (visité le 01/10/2018)
[3] https ://www.dropbox.com/ (visité le 01/10/2018)

La plate-forme (par exemple, Google App Engine [4], Heroku [5]) peut être un environnement de développement intégré, un serveur web ou un serveur d'application.

Infrastructure en tant que service (Infrastructure as a Service (IaaS))

Dans ce modèle, la ressource exposée est une infrastructure. Elle est offerte aux utilisateurs en tant que machines virtuelles. Cela permet aux fournisseurs de ce service (par exemple, Orange Cloudwatt [6], Amazon EC2 [7], Google Compute Engine [8]) de dimensionner les capacités exposées aux utilisateurs en fonction du besoin.

1.1.2 Modèles de déploiement du Cloud

Il existe essentiellement quatre modèles de déploiement du Cloud qui diffèrent selon la nature des utilisateurs des services.

Le Cloud privé

Les utilisateurs des services du Cloud privé font partie d'une même organisation. C'est cette organisation qui, dans la plupart des cas, détient et opère les ressources mais, dans le cas général, il est possible que les ressources soient détenues ou gérées par des tiers.

Le Cloud communautaire

Contrairement aux utilisateurs du Cloud privé, ceux du Cloud communautaire peuvent faire partie de différentes organisations à condition qu'elles appartiennent à la même communauté. Ainsi, les organisations doivent avoir les mêmes intérêts (par exemple, type d'applications, exigences de sécurité, règles et considérations de conformité).

Le Cloud public

Le Cloud public est le moins restrictif en comparaison avec les deux modèles précédents. En effet, il n'y a pas de contraintes sur ses utilisateurs qui peuvent faire partie du grand public. C'est le modèle suivi dans les offres de Cloud commerciales (par exemple, Dropbox, Orange Cloudwatt, Kafka on Heroku).

Le Cloud hybride

Ce modèle est une composition de plusieurs Clouds qui ont des modèles distincts (*c.-à-d.*, privés, communautaires ou publics). Un exemple d'usage typique de ce modèle est une composition d'un Cloud privé avec un Cloud public pour les entreprises qui nécessitent des ressources qui dépassent les capacités de leur Cloud.

[4] https ://cloud.google.com/appengine/ (visité le 01/10/2018)
[5] https ://www.heroku.com/ (visité le 01/10/2018)
[6] https ://www.cloudwatt.com/en/ (visité le 01/10/2018)
[7] https ://aws.amazon.com/fr/ec2/ (visité le 01/10/2018)
[8] https ://cloud.google.com/compute/ (visité le 01/10/2018)

1.1.3 Importance de l'économie d'échelle dans le succès du Cloud

Le Cloud a eu un grand succès. Cela se manifeste dans le chiffre d'affaires de son marché qui a atteint 180 milliards de dollars en 2017 [Synergy 2018]. Ce succès n'est pas uniquement le fruit des avantages techniques du Cloud mais est également associé à ses avantages économiques. En plus de la flexibilité qu'offre le Cloud à ses utilisateurs avec le modèle "Payer à l'usage", il offre ses services à des prix réduits par rapport au temps d'usage. Cela est réalisé par la mise en place d'une économie d'échelle. En effet, les fournisseurs du Cloud regroupent leurs ressources dans des centres de données gigantesques pour réduire les coûts de leur maintenance et simplifier leur gestion. Cependant, le modèle des centres de données gigantesques et centralisés fait que le Cloud ne soit pas adapté pour héberger les applications de nouvelle génération.

1.2 Défi des applications de nouvelle génération

Les applications de nouvelle génération ont de grandes exigences en latence et en bande passante qu'un hébergement dans une infrastructure Cloud classique ne peut pas satisfaire.

1.2.1 Applications de nouvelle génération et leurs exigences

Les applications de nouvelle génération ont un champ d'application très large qui concerne non seulement l'industrie mais qui touche aussi à notre mode de vie et nos relations humaines. Il couvre le transport, le divertissement, la santé, l'éducation et la culture. Une grande partie de ces applications est possible grâce à l'Internet des objets (IdO). Son principe consiste à connecter des objets (par exemple, capteurs, actionneurs) à Internet afin qu'ils puissent communiquer entre eux-mêmes et avec d'autres services exposés sur Internet. Cependant, pour réussir la mise en œuvre des différentes applications, il faut satisfaire leurs grandes exigences en latence et en bande passante illustrées dans la table 1.1. Dans les paragraphes suivants, nous exposons des exemples d'applications ainsi que leurs exigences comme décrit dans [Parvez et al. 2018].

Automatisation de l'industrie

À l'aide des objet connectés, il est possible de mettre en place une usine intelligente [Erbe 2004] où les employés et les machines accèdent à des informations détaillées sur l'environnement du travail (par exemple, emplacement du matériel, états des produits, avancement de la production...). Cela permet d'automatiser la production [Åkerberg et al. 2011] et de réduire l'intervention humaine. Cela permet également d'organiser le travail d'une manière plus efficace et de réduire les dépenses énergétiques [Rogers 2014]. Cependant, l'automatisation nécessite d'assurer une communication entre les capteurs et les unités de contrôle avec une latence entre 0.25 ms et 10 ms et une bande passante de l'ordre de 1 Mbps.

Réseaux électriques intelligents

Les réseaux électriques intelligents [Fang et al. 2012] sont une réponse à la difficulté du stockage de l'énergie électrique. Ils assurent une distribution efficace en s'appuyant sur des informations concernant sa production et sa consommation. Ces informations sont collectées à l'aide de compteurs connectés installés dans les stations de production et chez les clients (qui peuvent être également des producteurs d'énergie). L'objectif est d'adapter la production et l'acheminement de l'énergie en temps réel afin de limiter les pertes et les coupures. Pour cela, les réseaux électriques intelligents nécessitent une latence entre 1 ms et 20 ms et une bande passante entre 10 kbps et 100 kbps.

Systèmes de transport intelligents

Il est possible de développer les systèmes de transport en exploitant des informations sur les éléments qui les constituent (par exemple, véhicules, signaux de signalisation, usagers...). Par exemple, dans [Zhang et al. 2013], les auteurs proposent un algorithme qui s'appuie sur des informations remontées par des capteurs de passage de véhicules pour contrôler le trafic dans une autoroute. L'objectif est de maximiser le débit du trafic et d'éviter les embouteillages. Dans [Campolo et al. 2017], les auteurs étudient la capacité des réseaux mobiles "Long-Term Evolution" à assurer une communication qui permet d'échanger la position, la vitesse et l'accélération entre les voitures afin de réaliser un convoi. [Olaverri-Monreal et al. 2010] propose un système d'observation via les véhicules (See-Through System) pour échanger des vidéos entre leurs conducteurs afin de faciliter le dépassement des véhicules dont la longueur peut gêner la vision. Cependant, la mise en œuvre de telles applications nécessite une latence entre 10 ms et 100 ms et une bande passante entre 10 Mbps et 700 Mbps.

Les jeux sérieux

L'objectif des jeux sérieux ne se limite pas au divertissement. Il couvre aussi l'éducation, le soin, l'entraînement, la communication,... Grâce à l'IdO ces jeux simulent des situations réelles qui peuvent ne pas être accessibles dans la réalité pour des questions de coûts financiers, temps ou sécurité... Par exemple, SnowWorld [Hoffman et al. 2014] immerge des patients brûlés dans un monde de glace pour les aider à supporter les douleurs du traitement. Pour réussir la simulation, de telles applications nécessitent une latence de l'ordre de 1 ms et une bande passante de l'ordre de 1 Gbps.

Santé

L'IdO a plusieurs applications dans le domaine de la santé. Par exemple, grâce à des objets connectés, il est possible de surveiller la santé des patients alors qu'ils sont chez eux [Pang 2013]. Ainsi, les moyens disponibles aux hôpitaux seront consacrés aux cas les plus graves. En plus, les patients n'ont pas à se déplacer pour effectuer des contrôles médicaux routiniers. La télé-chirurgie [Marescaux et al. 2006] est une autre application prometteuse de l'IdO qui permettra à des chirurgiens d'opérer leurs malades à distance.

Par conséquent, le manque d'expertise dans les petits établissements de santé ou dans les zones rurales peut être surmonté. À noter que de telles applications nécessitent une latence entre 1 ms et 10 ms en plus d'une bande passante de l'ordre de 100 Mbps.

Education et culture

L'éducation peut profiter de l'IdO pour enrichir les cours offerts aux étudiants par des simulations. À l'aide d'équipements qui génèrent un ressenti tactile, il est possible de réaliser des maquettes virtuelles sur lesquelles les étudiants peuvent s'entraîner avant de travailler dans un contexte réel. [Stone 2001] propose un tel équipement. Il se compose d'actionneurs pneumatiques qui permettent de simuler un retour tactile quand l'utilisateur touche un objet virtuel. En plus, grâce à l'IdO, il est possible d'enrichir les visites culturelles des touristes. Par exemple, [Boletsis et al. 2018] proposent un guide audio pour une visite interactive des sites touristiques. Pour offrir un service de qualité, de telles applications nécessitent une latence entre 5 ms et 10 ms et une bande passante de l'ordre de 1 Gbps.

Application	**Exigence en latence**	**Exigence en bande passante**
Santé	1-10 ms	100 Mbps
Systèmes de transport intelligents	10-100 ms	10-700 Mbps
Automatisation de l'industrie	0.25-10 ms	1 Mbps
Réseaux électriques intelligents	1-20 ms	10-1500 Kbps
Education et culture	5-10 ms	1 Gbps
Jeux sérieux	1 ms	1 Gbps

© 2018 IEEE

Table 1.1 – Exigences en latence et en bande passante des applications de nouvelle génération [Parvez et al. 2018]

1.2.2 Limites du Cloud Computing

La latence pour joindre un serveur du Cloud peut varier de plusieurs millisecondes à plusieurs secondes [Ha et al. 2013]. Une étude de cas présentée dans [Choy et al. 2014], a montré que la latence d'accès au Cloud dépasse 80 ms pour 25% des utilisateurs et 160 ms pour 10% des utilisateurs. Les auteurs ont subdivisé cette latence en quatre composantes. La première est la latence entre l'équipement de l'utilisateur et le premier routeur connecté à Internet. Sa valeur dépasse les 10 ms pour 75% des utilisateurs dans le cas où le réseau n'est pas surchargé. Dans le cas contraire, sa valeur moyenne dépasse 40 ms. Cette latence dépend fortement de la technologie d'accès utilisée. La deuxième composante est la latence introduite par le fournisseur permettant à l'utilisateur d'accéder à Internet. La troisième composante est la latence introduite par les fournisseurs intermédiaires d'accès à Internet qui permettent de rejoindre le serveur frontal du Cloud. Enfin, la dernière composante est la latence entre le serveur frontal du Cloud et le serveur hébergeant

le service de l'utilisateur. Contrairement aux latences précédentes, cette dernière n'est pas introduite par le réseau d'accès au Cloud mais par son réseau interne. Sa valeur est inférieure à 1 ms dans les centres de données modernes [Rumble et al. 2011]. À noter qu'en plus du délai de traitement, le serveur hébergeant le service de l'utilisateur introduit une latence supplémentaire car les processeurs ne suivent pas la cadence du réseau. La nature de ces latences confirme que la latence globale pour joindre le Cloud est imprévisible et élevée par rapport aux besoins des applications de nouvelle génération comme cela a été constaté dans [Zhang et al. 2015].

En plus, le réseau qui permet l'accès au Cloud ne dispose pas d'une bande passante suffisante pour les besoins des applications de nouvelle génération. Ce réseau a été conçu pour un usage traditionnel d'Internet où les requêtes des utilisateurs sont limitées en nombre et en fréquence. Ce mode d'utilisation a été complètement bouleversé par la prolifération des objets connectés. Ces objets ont augmenté considérablement le nombre d'utilisateurs d'Internet qui ne se limitent plus aux êtres humains mais qui peuvent être aussi des objets. Minoli *et al.* estiment que le nombre de ces objets sera de l'ordre de 30 milliards en 2020 [Minoli et al. 2017]. En contrepartie, le réseau n'a pas évolué aussi rapidement pour s'adapter à ce changement. Actuellement, il n'est même pas en mesure d'assurer la diffusion des événements qui ont un grand nombre de spectateurs. Par exemple, lors d'un match de boxe diffusé récemment à Las Vegas aux Etats Unis, il y a eu une congestion au niveau du réseau quand le nombre de spectateurs a atteint un pic [ESPN 2017]. En plus, les objets connectés ne sont pas des utilisateurs traditionnels d'Internet qui se limitent à accéder aux données mais qui les créent aussi. Rumble *et al.* estiment que la taille de ces données atteindra 500 Zo en 2019 alors que le réseau qui permet d'accéder au Cloud ne permet de transporter que 10.4 Zo [Rumble et al. 2011].

Pour conclure, le Cloud n'est pas en mesure d'assurer ni la latence ni le débit requis par les applications de nouvelle génération.

1.3 L'Edge Computing : une réponse aux exigences des applications de nouvelle génération

Envoyer toutes les requêtes des utilisateurs à un Cloud centralisé et leur retourner ensuite les résultats des traitements entrave la satisfaction des exigences de latence et de bande passante des applications de nouvelle génération. Pour cette raison, les experts du monde académique et industriel recommandent de ne pas se limiter au modèle du Cloud basé sur des infrastructures gigantesques et centralisées. Ils proposent de s'appuyer sur des infrastructures massivement distribuées qui peuvent être déployées à proximité des utilisateurs finaux. Ce modèle d'infrastructure permet de réduire le temps de réponse. En plus, il augmente le débit car il répartit la charge sur le réseau. La table 1.2 montre l'impact de la distance entre l'utilisateur et le serveur hôte sur le temps de parcours (RTT) et le débit. Ces tests ont été réalisés avec TCP mais le reste des conditions expérimentales qui peuvent avoir un impact sur ces mesures (par exemple, la nature des liens, la topologie du réseau, le nombre de sauts) n'ont pas été renseignées.

Distance	RTT	Perte de paquets	Débit	Temps de téléchargement de 4 GB
Locale : <100 miles	1.6 ms	0.6%	44 Mbps	12 min
Régionale : 500–100 miles	16 ms	0.7%	4 Mbps	2.2 h
Cross-continentale : 3000 miles	48 ms	1.0%	1 Mbps	8.2 h
Multi-continentale : 6000 miles	96 ms	1.4%	0.4 Mbps	20 h

TABLE 1.2 – Impact de la distance entre l'utilisateur et le serveur sur le temps de parcours (RTT), taux de perte de paquets, le débit et le temps de téléchargement [LEIGHTON 2009]

1.3.1 Présentation du Edge Computing

Plusieurs paradigmes qui étendent le Cloud pour s'appuyer sur des infrastructures distribuées ont été proposés. Nous citons à titre d'exemple : le Fog Computing [BONOMI et al. 2012], le Mobile Cloud Computing [DINH et al. 2013], le mobile Edge Computing [AHMED et al. 2016] et l'Edge Computing [SHI et al. 2016]. Ces paradigmes considèrent des niveaux différents de distribution des ressources. Dans notre étude, nous considérons l'Edge Computing qui couvre l'ensemble de ces niveaux.

1.3.2 Infrastructure Edge

A titre d'exemple, l'infrastructure Edge [CONFAIS et al. 2016] que nous considérons dans la suite, est construite selon le point de vue d'un opérateur de télécommunications. Elle est subdivisée en quatre niveaux de ressources représentés sur la figure 1.1 [ABDERRAHIM et al. 2017].

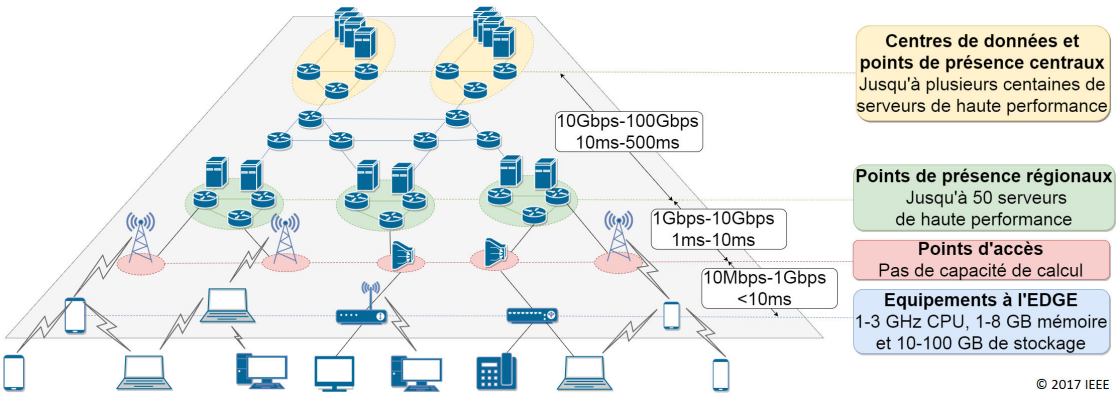

FIGURE 1.1 – Infrastructure cible

Premier niveau

Au niveau le plus haut, quelques points de présence (PoPs) centraux (*c.-à-d.*, nationaux ou internationaux) offrent des capacités massives en termes de calcul, stockage et réseau. Ils peuvent contenir des centaines de serveurs d'une haute performance ainsi que des équipements réseau pour interconnecter les serveurs ou pour les connecter aux sites externes. Chaque serveur a des capacités minimales de l'ordre de 32 à 64 Cœurs de 2 GHz, 64 à 128 Go de mémoire et plus de 1 To de stockage. Il est destiné à héberger des services informatiques de gestion (par exemple, service de facturation), services de gestion réseau (par exemple, Evolved Packet Core functions), ou des services de tiers.

Deuxième niveau

À un niveau inférieur, des dizaines de PoPs régionaux hébergent des serveurs similaires à ceux des PoPs centraux en termes de performance. Cependant, le nombre de serveurs dans chaque PoP régional ne dépasse pas cinquante. Étant donné que ces PoPs sont plus proches des utilisateurs finaux (ils se trouvent dans un rayon de 200 km des utilisateurs), ils peuvent accueillir des services sensibles à la latence (par exemple, passerelles résidentielles) ou services générant un trafic de données important (par exemple, réseau de diffusion de contenu) afin d'améliorer la qualité de l'expérience de l'utilisateur (QoE). Ils sont connectés aux PoPs centraux grâce à des liens réseau en fibre optique qui ont une haute résilience et qui assurent des latences de quelques centaines de millisecondes.

Troisième niveau

Le niveau des nœuds d'accès (AN) avec quelques milliers d'éléments n'offre ni des capacités de calcul ni de stockage. Il s'agit des liens optiques et des points d'accès radio dont le rôle est limité à assurer l'accès des utilisateurs aux services hébergés sur l'infrastructure. Ils sont conçus pour être les plus légers possible et peuvent donc être installés sur les sites loués par l'opérateur d'infrastructure (par exemple, zones publiques ou privées). Même le traitement du signal intelligent, fonction traditionnellement hébergée sur l'AN, peut être hébergé sur un plus haut niveau (comme les PoPs régionaux) grâce au CloudRAN [Checko et al. 2015]. Cette technologie permet de virtualiser la fonction du traitement du signal intelligent. Ainsi, elle permet d'adapter les capacités des ressources qui l'hébergent en fonction du besoin.

Quatrième niveau

Au niveau le plus bas, l'opérateur de l'infrastructure peut s'appuyer sur des ressources de calcul et de réseau parmi quelques millions de ressources Edge (ED) telles que les mobiles personnels et les passerelles résidentielles. Ces ressources ont une capacité limitée avec des fréquences de CPU de l'ordre de 1 GHz, une mémoire de l'ordre de 1 GB et une capacité de stockage jusqu'à des dizaines ou des centaines de GB. En comparaison avec les ressources des autres niveaux de l'infrastructure, celles-ci sont destinées à servir un nombre beaucoup plus restreint d'utilisateurs. Pour cela, leur temps d'accès disque et leur

performance en termes d'entrée/sortie (I/O) sont bien inférieures. De plus, cette capacité est d'abord dédiée aux besoins du propriétaire de l'ED. L'utilisation des ressources restantes est principalement opportuniste. L'opérateur peut avoir un contrôle limité sur la disponibilité de la ressource et l'établissement de la connectivité. Le nombre de ED exploitables varie aléatoirement. De telles ressources peuvent être invoquées pour des utilisations temporaires ou événementielles. Certaines, comme les mobiles personnels, sont volatiles. Elles peuvent changer fréquemment d'emplacement et être momentanément injoignables.

1.3.3 Caractéristiques de l'infrastructure Edge

La description de l'infrastructure Edge dans la section 1.3 montre qu'elle diffère significativement d'une infrastructure Cloud traditionnelle. Elle a trois caractéristiques intrinsèques qu'il faut prendre en considération lors de sa gestion :

Distribution massive

Alors que les ressources de l'infrastructure Cloud traditionnelle occupent un nombre très limité de sites, les ressources de l'infrastructure Edge sont réparties à travers un grand nombre de sites. Les plus performantes d'entre elles occupent des dizaines de PoPs. Les ressources potentielles à la périphérie du réseau est de l'ordre de millions. Toutes peuvent appartenir à différents domaines administratifs. La distance qui les sépare peut aller jusqu'à des centaines de kilomètres au plus haut niveau de l'infrastructure. La latence introduite par les liens les reliant peut atteindre 500 ms et leurs bandes passantes peuvent être limitées à 10 Mbps.

Hétérogénéité

L'infrastructure est composée de plusieurs ressources hétérogènes : serveurs de stockage, serveurs de calcul, routeurs, commutateurs génériques, passerelles résidentielles, équipements des utilisateurs, plates-formes logicielles, des liens filaires (par exemple, cuivre, fibre optique) ou sans fil (par exemple, micro-ondes ou Wi-Fi). Toutes ces ressources ont des caractéristiques différentes en termes de capacité (par exemple, des serveurs de haute performance, des équipements utilisateurs avec des capacités modestes), de fiabilité (dédiées à l'infrastructure comme les routeurs dans les PoPs ou provisoires comme un terminal utilisateur) et d'utilisation (par exemple, calcul, réseau, stokage). La virtualisation introduit un autre niveau d'hétérogénéité puisque les ressources exploitées peuvent être physiques ou virtuelles. Cela concerne les ressources de stockage ou de calcul ainsi que les ressources réseau qui peuvent être virtualisées grâce à la virtualisation des fonctions réseau (NFV) [Mijumbi et al. 2016].

Dynamicité

Les ressources d'une infrastructure Edge peuvent être très dynamiques. Cela peut être constaté au niveau des tâches qui sont fréquemment instanciées avec des cycles de vie

courts et qui sont migrées pour s'adapter par exemple à une augmentation de charge, panne d'équipements ou optimisation de l'utilisation d'énergie. Cela peut être constaté également au niveau le plus bas de l'infrastructure où les ED peuvent changer d'emplacement ou quitter et rejoindre le réseau en performance à cause des déconnexions fréquentes.

1.4 Gestionnaire de ressources pour l'infrastructure Edge

Afin de satisfaire les attentes des opérateurs et des utilisateurs des infrastructures Edge, des services de gestion ayant des capacités similaires à celles qui ont permis le succès du Cloud doivent être conçus. Ils doivent, en premier lieu, permettre à l'opérateur d'agréger, d'exposer et de superviser les ressources d'une infrastructure Edge et doivent, en second lieu, permettre à des tiers de déployer leurs services sur ces ressources.

Cependant, la réutilisation des solutions de gestion destinées au Cloud telles qu'OpenStack [9] ou OpenMANO [10] n'est pas évidente car la plupart d'entre elles ont été conçues pour des architectures de ressources centralisées [Lebre et al. 2017]. Autrement dit, ces solutions ne prennent pas en considération les spécificités de l'infrastructure Edge (*c.-à-d.*, distribution massive, hétérogénéité, dynamicité). Par ailleurs, des solutions de gestion opérationnelles exploitant les bénéfices de l'infrastructure Edge sont apparues telles que Akamai+Cloudlet [11] ou Amazon Lambda@Edge [12]. Cependant, ce type de solutions reste limité à des applications spécifiques (par exemple, load balancer).

Proposer un canevas logiciel pour l'ensemble de fonctionnalités que doit proposer un gestionnaire de l'infrastructure Edge est un travail considérable qui demande des expertises dans différents domaines. Dans cette thèse, nous nous intéressons d'une manière plus spécifique à la supervision qui est une fonction clé de gestion. La supervision est utilisée, par exemple, pour garantir la qualité de service, détecter les pannes ou assurer la sécurité. Elle est d'autant plus importante dans le cas d'infrastructure Edge où les ressources sont plus hétérogènes, réparties et dynamiques.

Néanmoins, maintenir une fonction de supervision peut être une tâche très complexe et coûteuse. D'une part, il est nécessaire de l'adapter aux différents types de ressources observées indépendamment de leur technologie, de remonter leur état à une fréquence élevée, voire en temps réel, et d'assurer des propriétés de robustesse comme le passage à l'échelle et la résilience aux pannes. D'autre part, l'empreinte de cette fonction risque d'être non négligeable sur l'infrastructure. À titre d'exemple, la supervision de l'infrastructure de Twitter génère 2.8 milliards de mesures par minute [Asta 2016]. Cette empreinte ne se limite pas au calcul et concerne également le réseau étant donné l'aspect massivement distribué de l'infrastructure Edge.

Pour ces raisons, de nombreux utilisateurs optent pour des solutions qui offrent la supervision en tant que service utilitaire [Romano et al. 2011] comme CloudWatch [13],

[9] https ://www.openstack.org/ (visité le 01/10/2018)
[10] https ://osm.etsi.org/ (visité le 01/10/2018)
[11] http ://cloudlets.akamai.com (visité le 01/10/2018)
[12] https ://aws.amazon.com/lambda/edge/ (visité le 01/10/2018)
[13] https ://aws.amazon.com/cloudwatch/ (visité le 01/10/2018)

Datadog [14], New Relic [15] ... Ces services dispensent leurs utilisateurs de la complexité du maintien de la fonction de supervision. Cependant, aucun de ces services n'a été conçu en prenant en considération les spécificités de l'infrastructure Edge comme nous allons l'aborder dans ce manuscrit.

Un service de supervision d'une infrastructure Edge doit être holistique. En d'autres termes, comme l'illustre la figure 1.2, il doit assurer l'observation de toutes les ressources de l'infrastructure Edge qui peuvent appartenir à l'opérateur (par exemple, serveurs, routeurs, liens réseau), aux fournisseurs des ressources Edge à la périphérie du réseau (par exemple, les passerelles résidentielles, les smartphones, les ordinateurs portables), aux fournisseurs de services (par exemple, contenu multimédia, sites Web) et aux utilisateurs de services (par exemple, des machines virtuelles, des fonctions de réseau virtuel, des capteurs de température).

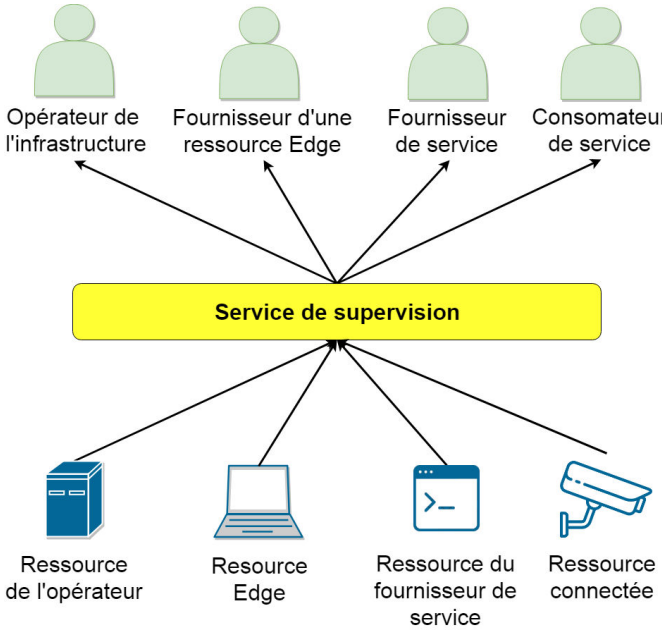

FIGURE 1.2 – Écosystème du service de supervision holistique

1.5 Conclusion

Les applications de nouvelle génération ont de fortes exigences en latence et en bande passante. Le Cloud Computing n'est pas en mesure de satisfaire ces exigences car il s'appuie sur des infrastructures centralisées. En s'appuyant sur des infrastructures massivement distribuées, l'Edge Computing cherche à adapter le Cloud aux besoins de ces applications. Cependant, la gestion, et en particulier, la supervision de telles infrastructures constituent un défi.

[14] https ://www.datadoghq.com/ (visité le 01/10/2018)
[15] https ://www.newrelic.com/ (visité le 01/10/2018)

Dans le chapitre suivant, nous allons spécifier les propriétés qui doivent être satisfaites pour réussir la supervision de l'infrastructure Edge. En plus, nous allons évaluer les solutions de supervision existantes par rapport à ces propriétés.

2

État de l'art

Sommaire

Dans ce chapitre, nous nous concentrons sur la solution de supervision sur laquelle doit s'appuyer le service à offrir aux tenants de l'infrastructure Edge. Nous vérifions si les solutions de supervision existantes sont adaptées à notre besoin. Pour cela, nous commençons par définir les exigences à satisfaire afin d'adapter la solution aux caractéristiques de l'infrastructure Edge. Ensuite, nous évaluons qualitativement les solutions de supervision existantes par rapport à ces exigences. Enfin, nous introduisons notre contribution.

2.1 Spécifications de la solution de supervision

Nous définissons les exigences que doit satisfaire la solution de supervision en considérant les caractéristiques de l'infrastructure Edge. Nous formalisons ces exigences en tant que propriétés fonctionnelles et propriétés non fonctionnelles.

2.1.1 Propriétés fonctionnelles

La solution de supervision doit assurer essentiellement trois fonctions :

Observation

La solution de supervision doit pouvoir accéder aux états des ressources de l'infrastructure Edge malgré leur hétérogénéité. Il y a deux situations qui se présentent : soit la ressource supervisée expose son état uniquement à distance (via une interface REST par exemple), soit elle permet d'héberger localement une fonction d'observation (souvent appelée sonde ou capteur).

Deux types de fonctions d'observation sont généralement distingués. En effet, l'observation est dite "active" lorsqu'elle nécessite l'injection de paquets supplémentaires sur le réseau. Par exemple, l'outil *Ping* permet d'observer l'état du réseau via l'envoie et la réception de paquets ICMP echo request et ICMP echo reply. Lorsqu'elle n'injecte pas des paquets supplémentaires sur le réseau, l'observation est dite passive. Par exemple, les renifleurs tel que *Wireshark* permettent de récupérer les données échangées qui circulent sur le réseau sans apporter des modifications. L'observation active réduit la charge introduite sur l'infrastructure. Cependant, il peut être plus coûteux. Par exemple, pour mesurer d'une manière passive la latence du réseau, il faut synchroniser plusieurs sondes [Ping 2003].

Deux modes de communication sont souvent utilisés pour remonter les mesures : "push" et "pull". Dans le mode "pull", la seconde envoie les mesures en réponse à une demande. Dans le mode "push", la sonde envoie les mesures, généralement d'une manière périodique, à une destination sans que la dernière fasse une demande au préalable.

Traitement

Les mesures observées représentent souvent des données brutes qui ne correspondent pas forcément aux états requis par les utilisateurs. Pour cette raison, elles doivent être traitées. Le traitement peut consister à l'agrégation des mesures remontées par un seul capteur (par exemple, pour ajuster la fenêtre d'échantillonnage, pour transformer les mesures en des événements ou alarmes) ou à la composition des mesures remontées par plusieurs capteurs.

Exposition

Les résultats de traitement doivent être exposés aux utilisateurs concernés. Ces derniers peuvent être des êtres humains ou des applications. Pour cela, le service de supervision

doit supporter différents types d'interfaces (par exemple, interface graphique, lignes de commandes, interface de programmation (API)) et de notifications (par exemple, SMS, courriel, interruption système).

2.1.2 Propriétés non fonctionnelles

En plus des propriétés fonctionnelles, la solution de supervision doit considérer les spécificités de l'infrastructure Edge. À cette fin, elle doit satisfaire les propriétés suivantes :

Compatibilité vis-à-vis des ressources

Les ressources de l'infrastructure Edge sont très hétérogènes. Leurs capacités de traitement peuvent être considérables (par exemple, les serveurs de hautes performances) ou modestes (par exemple, les équipements des utilisateurs). La solution de supervision doit être compatible avec toutes les ressources indépendamment de leurs capacités.

Passage à l'échelle

La solution de supervision doit être en mesure d'observer un grand nombre de ressources et d'exposer leurs états aux utilisateurs. Elle doit pouvoir s'adapter à une augmentation brutale de charge sans que cela n'impacte ses performances. Dans certains cas, l'augmentation des capacités des ressources hôtes du service (*c.-à-d.*, passage à l'échelle vertical) peut ne pas être possible (par exemple, la ressource hôte est une machine physique). Par conséquent, la solution de supervision doit supporter le passage à l'échelle horizontal. Autrement dit, elle doit pouvoir exploiter une multitude de ressources hôtes.

Proximité vis-à-vis des utilisateurs

Le service de supervision doit assurer le temps de réponse requis par les utilisateurs indépendamment de l'emplacement des ressources supervisées. Cela est particulièrement difficile dans une infrastructure massivement distribuée. Pour cela le service de supervision doit permettre un déploiement à proximité des ressources observées et des utilisateurs intéressés par leurs états.

Résilience aux pannes/volatilité des serveurs

Les serveurs de l'infrastructure sont toujours susceptibles d'être hors service en raison de pannes. Ces pannes peuvent provenir de différentes sources : les machines physiques, les machines virtuelles, les applications hébergées ou les tâches exécutées [Prathiba et al. 2017]. En plus, la mobilité des ressources à la périphérie du réseau comme les téléphones personnels peut interrompre les services hébergés sur l'infrastructure. Il faut donc que la solution de supervision soit résiliente à ces événements.

Résilience aux changements/pannes du réseau

Étant donnée sa topologie massivement distribuée, l'infrastructure Edge est particulièrement vulnérable aux pannes réseau vu qu'elle est massivement distribuée. Ces pannes risquent d'affecter la solution de supervision qui s'appuie sur le réseau pour observer les ressources distantes. Le service de supervision doit donc être résilient aux pannes réseau. Il doit, par exemple, assurer la retransmission des données importantes suite aux pannes.

2.2 Une approche qualitative d'évaluation

Le nombre de solutions qui ont été développées pour superviser les infrastructures IT et réseau est très grand [Ward et al. 2014]. En plus, ces solutions ne sont pas toutes à code source ouvert et n'ont pas toutes une documentation détaillée. Pour ces raisons, s'appuyer sur une approche empirique pour les évaluer par rapport aux propriétés spécifiées n'était pas une tâche envisageable. En effet, il faut concevoir un grand nombre de tests pour couvrir toutes les combinaisons de déploiements possibles, nature de mesures, charge de mesures, variation du nombre d'utilisateurs ou de ressources supervisées, occurrences de pannes . . . tout en prenant en considération la distribution massive, la large hétérogénéité et la grande dynamicité de l'infrastructure Edge. Ainsi, nous avons opté pour une évaluation qualitative. Dans cette étude, nous avons classé les solutions existantes selon leurs architectures logicielles dans neuf catégories. Ces catégories diffèrent les unes des autres selon le modèle architectural (*c.-à-d.*, centralisé, hiérarchique ou pair-à-pair) et la décomposition fonctionnelle (*c.-à-d.*, aucune, élémentaire, avancée). La figure 2.1 présente les neuf catégories. Dans ce qui suit, nous discutons comment chacune répond aux propriétés spécifiées dans la section précédente.

2.2.1 Décomposition fonctionnelle

Nous identifions trois décompositions fonctionnelles possibles pour les architectures des solutions de supervision : aucune décomposition, décomposition élémentaire et décomposition avancée. Elles correspondent aux lignes de la figure 2.1. Elles répondent différemment à la *résilience aux changements/pannes du réseau*, la *compatibilité vis-à-vis des ressources* et la *réduction de l'empreinte réseau* comme l'illustre la table 2.1.

		Satisfaction des propriétés requises	
		Résilience aux changements et pannes du réseau	Compatibilité vis-à-vis des ressources
Décomposi -tions fonc -tionnelles	Aucune	-	-
	Élémentaire	+	-
	Avancée	+	+

Table 2.1 – Satisfaction des propriétés spécifiées par les différentes décompositions fonctionnelles

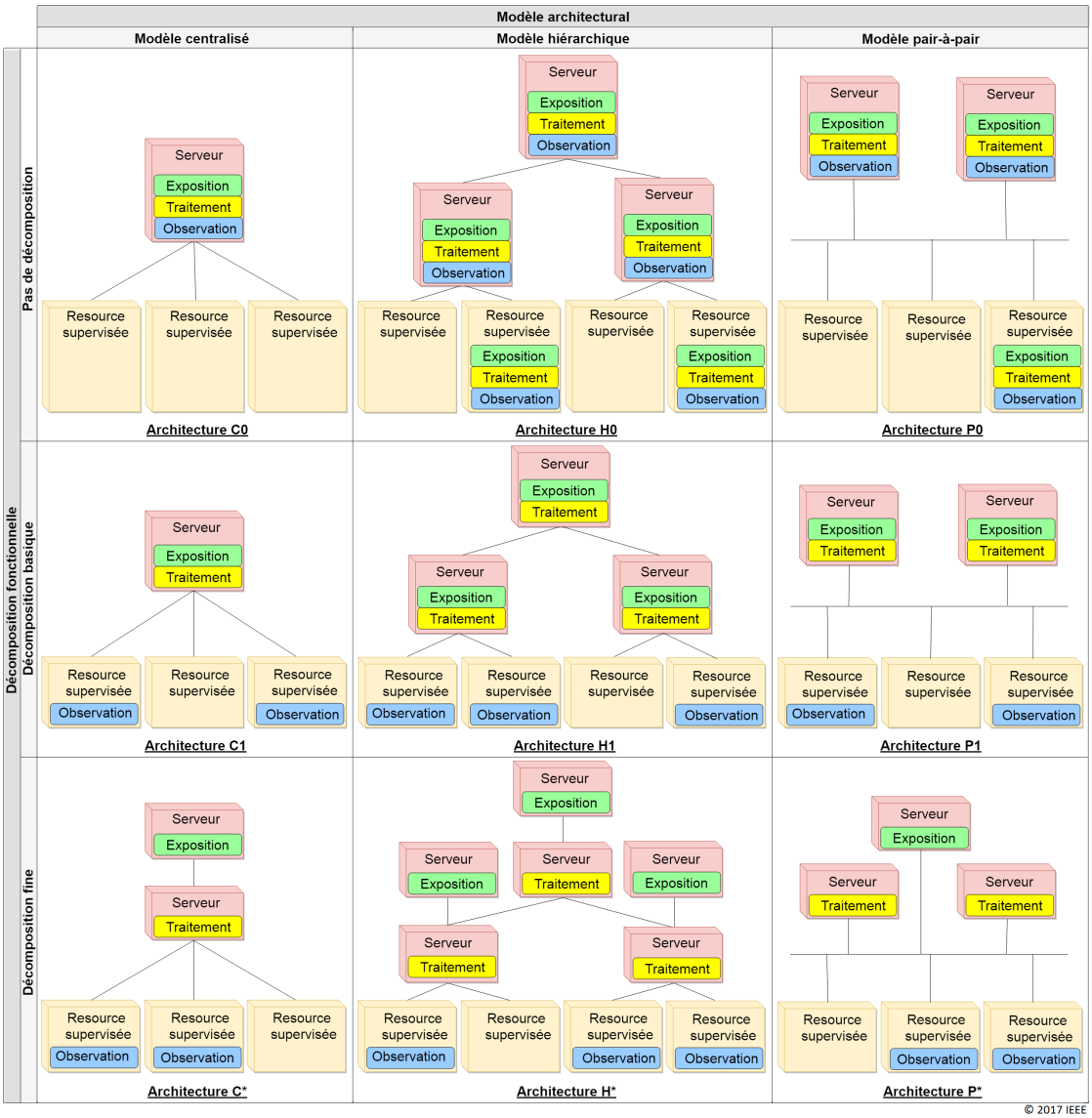

FIGURE 2.1 – Architectures principales pour une solution de supervision

Aucune décomposition

Dans cette catégorie d'architectures, les trois fonctions spécifiées précédemment sont assurées par un élément monolithique : il observe la ressource hôte (si nécessaire), traite les mesures observées et expose les résultats de traitement aux utilisateurs. Ainsi, pour superviser les états des ressources distantes, il s'appuie sur des sondes offertes par ces ressources.

Ce mode d'observation est typique pour les ressources qui n'autorisent pas l'hébergement d'une sonde externe pour observer leurs états. Par exemple, une ressource logicielle telle qu'une fonction de réseau virtuel propriétaire restreint sa supervision avec ses propres sondes pour des questions de sécurité. Les sondes contextuelles installées dans un centre de données pour observer les conditions environnementales (par exemple, température, humidité, consommation d'énergie) exposent généralement leurs mesures via une API et ne sont pas en mesure d'héberger une sonde spécifique.

Décomposition élémentaire

Dans cette catégorie d'architectures, la fonction d'observation peut être exécutée séparément du reste des fonctions (*c.-à-d.*, fonctions de traitement et d'exposition). Son déploiement sur la ressource supervisée est adapté aux ressources qui n'offrent pas de sondes et qui ont des capacités limitées et insatisfaisantes pour le déploiement de l'ensemble des fonctions de supervision.

Le déploiement d'une sonde sur la ressource supervisée donne plus de contrôle sur la fonction d'observation. Cela permet en particulier d'adapter cette fonction aux conditions du réseau : changer le protocole de communication (par exemple, utiliser TCP pour favoriser la sûreté de la communication et utiliser UDP pour favoriser sa continuité), changer le comportement suite à une déconnexion (par exemple, enregistrer dans une mémoire tampon les mesures non remontées ou abandonner l'envoi), changer le comportement suite au changement d'emplacement de la ressource supervisée ou un changement du réseau (continuer à envoyer les mesures vers la même destination ou les envoyer à une destination plus proche)... Ainsi, grâce à ce premier niveau de décomposition, il est possible d'améliorer la résilience aux changements/pannes du réseau.

Décomposition avancée

Dans cette catégorie, la décomposition fonctionnelle est maximale. Ainsi, chaque fonction élémentaire d'observation, traitement ou exposition peut être exécutée sur un serveur distant. Pour simplifier la figure, nous n'avons pas représenté les fonctions de traitement spécifiquement (par exemple, agrégation, transformation des mesures en événements, traitement des événements) dans la troisième ligne du tableau de la figure 2.1 mais nous avons représenté une fonction unique appelée traitement.

Dans cette catégorie d'architectures, la fonction d'observation peut être exécutée à distance sur la ressource supervisée. Ainsi, la *résilience aux changements/pannes réseau* est améliorée pour les raisons présentées précédemment. En plus, la décomposition

avancée permet de satisfaire la *compatibilité vis-à-vis des ressources* car elle apporte de la flexibilité en termes de déploiement sur les ressources hôtes de capacités hétérogènes. Ainsi, le service de supervision a une plus grande marge de manœuvre pour exploiter les capacités de toutes les ressources de l'infrastructure Edge.

2.2.2 Modèles architecturaux

En plus des différentes décompositions fonctionnelles, nous identifions trois modèles architecturaux : centralisé, hiérarchique et pair-à-pair. Ils sont représentés sur les colonnes de la figure 2.1. Ils répondent différemment à la *proximité vis-à-vis des utilisateurs*, à la *résilience aux pannes/disparition des serveurs* et au *passage à l'échelle* comme l'illustre la table 2.2.

		Satisfaction des propriétés requises		
		Résilience aux pannes/ volatilité des serveurs	Passage à l'échelle	Proximité vis-à-vis des utilisateurs
Modèle architectural	Centralisé	-	-	-
	Hiérarchique	-	+	+
	pair-à-pair	+	+	+

TABLE 2.2 – Satisfaction des propriétés spécifiées par les différentes décompositions fonctionnelles

Modèle centralisé

Ce modèle s'appuie sur une seule instance de chaque fonction de traitement et d'exposition. Il peut s'appuyer, par contre, sur différentes instances de fonctions d'observation. Cette catégorie d'architectures est représentée sur la première colonne du tableau de la figure 2.1.

Ce modèle architectural est le plus intuitif et le plus simple à réaliser. Cependant, étant donné que la multi-instanciation des fonctions d'observation et de traitement n'est pas possible, une solution ayant ce modèle architectural ne peut pas être déployée à proximité de tous les utilisateurs et de toutes les ressources supervisées en même temps dans un contexte d'infrastructure Edge massivement distribuée. Elle ne peut donc pas satisfaire la *proximité vis-à-vis des utilisateurs*. En plus, ce modèle est limité en termes de *passage à l'échelle*. En effet, vu que les éléments de traitement et d'exposition ne peuvent pas être multi-instanciés, le passage à l'échelle horizontal n'est pas supporté. Seul le passage à l'échelle vertical (augmentation des capacités des ressources hôtes) peut être mis en place. Enfin, la solution s'appuyant sur ce modèle n'est pas *résiliente aux pannes/volatilité des serveurs* car l'instance qui ne peut pas être multi-instanciée représente un point de défaillance critique.

Modèle hiérarchique

Le modèle hiérarchique n'impose pas de contrainte sur le nombre d'instances des fonctions. Cependant, il exige que les instances forment une structure arborescente. Autrement dit, chaque instance envoie un flux à une autre dans un niveau hiérarchique supérieur. L'instance de plus haut niveau dispose de la vue la plus globale de l'infrastructure et, par conséquent, elle est la plus adaptée pour assurer la fonction d'exposition. Cette catégorie d'architecture correspond à la deuxième colonne du tableau de la figure 2.1.

En permettant la multi-instanciation des fonctions de supervision, ce modèle permet de les répartir à proximité des ressources supervisées et des utilisateurs. Ainsi, il satisfait la *proximité vis-à-vis des utilisateurs*. En plus, la possibilité de multi-instancier des éléments permet d'assurer le *passage à l'échelle* horizontal. Cependant, ce modèle n'est pas *résilient aux pannes/volatilité* des serveurs. En effet, la maintenance des architectures hiérarchiques est complexe et coûteuse. Pour remplacer une ressource hôte en panne, il faut reconstruire l'organisation de l'arbre des instances. Cela nécessite un processus électoral pouvant introduire une charge et un délai non négligeables [Zhang et al. 2009]. Plus le niveau du nœud en panne ou disparu est élevé, plus le nombre des nœuds concernés par le processus électoral est grand et plus l'impact de la maintenance est important.

Modèle pair-à-pair

Ce dernier modèle d'architecture permet de multi-instancier les fonctions de supervision sans imposer une structure spécifique entre elles. N'importe quelle instance peut être connectée à une autre selon le modèle de communication pair-à-pair. Cette catégorie d'architectures est représentée sur la troisième colonne de la figure 2.1.

En permettant de multi-instancier les fonctions de supervision, ce modèle architectural retient les propriétés satisfaites par le modèle hiérarchique qui sont la *proximité vis-à-vis des utilisateurs* et le *passage à l'échelle*. De plus, l'absence de hiérarchie entre les fonctions dans cette architecture limite l'impacte de la panne [Mastroianni et al. 2008]. En effet, le système de supervision n'est pas coupé en deux sous-systèmes isolés lorsque un élément n'est pas fonctionnel. Ainsi, ce modèle architectural satisfait la *résilience à la volatilité/pannes des serveurs*.

2.3 Évaluation des solutions existantes

En combinant les différents modèles architecturaux et décompositions fonctionnelles, nous définissons les neuf catégories d'architectures illustrées par la figure 2.1. L'évaluation de chacune par rapport aux propriétés requises dépend de son modèle architectural et de sa décomposition fonctionnelle comme cela a été détaillé dans la section précédente. Pour évaluer qualitativement les principales solutions de la littérature, nous les classons selon ces neuf catégories.

2.3.1 Aucune décomposition fonctionnelle

Les solutions n'ayant aucune décomposition fonctionnelle n'utilisent pas leurs propres sondes pour superviser les ressources distantes mais elles utilisent des sondes offertes par les ressources.

Modèle centralisé

Les solutions qui ont été conçues selon le modèle centralisé et qui n'ont aucune décomposition fonctionnelle s'appuient sur un seul élément qui assure les fonctions d'observation, traitement et d'exposition. Nous appelons cette architecture C0, sur la figure 2.1.

Near Field Monitoring [Suneja et al. 2014] (NFM) est une solution destinée à la supervision des machines virtuelles. Elle s'appuie sur des techniques d'introspection. Comme illustré par la figure 2.2, elle fait appel à l'API de l'hyperviseur pour accéder à des captures (ou frames) de la mémoire disque et la mémoire vive de la machine virtuelle supervisée. Les informations disponibles au niveau de la mémoire disque lui permettent de déterminer le système d'exploitation de la machine virtuelle ainsi que sa configuration. En utilisant ces informations, NFM détermine les états des machines virtuelles supervisées. Il enregistre ces états dans une base de données centrale et les expose via son API aux applications qui sont éligibles à y accéder. À noter que les techniques d'introspection nécessitent d'analyser le noyau du système d'exploitation ce qui introduit un temps considérable à la supervision. En plus, l'application de cette technique peut être inefficace voire impossible pour les systèmes d'exploitation qui sont propriétaires ou n'offrent pas une documentation suffisamment détaillée. Enfin, cette technique expose les données disponibles au niveau des machines virtuelles ce qui risque d'introduire des failles de sécurité.

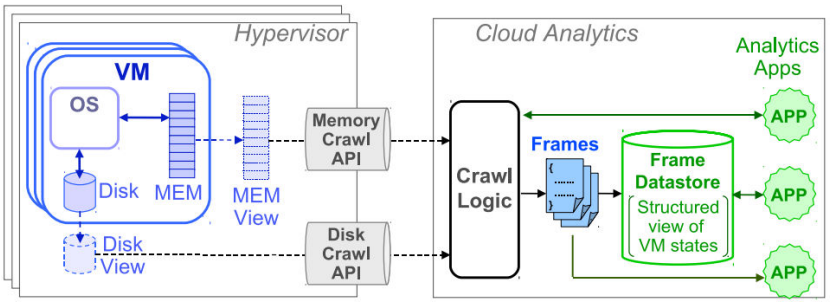

Figure 2.2 – Architecture de NFM [Suneja et al. 2014]

La supervision sans déployer une sonde sur la ressource supervisée présente plusieurs avantages. Par exemple, cela permet d'éviter d'introduire des vulnérabilités sur cette ressource. Cependant, il faut que la ressource permette l'observation de son état d'une manière compatible avec la solution de supervision. Afin de standardiser ce mode de supervision et lui offrir ainsi plus d'interopérabilité, des protocoles ont été proposés par la communauté réseau :

- **Simple Network Management Protocol** (SNMP) [Stallings 1998] a été développé en 1988 et est ainsi largement supporté par les équipements réseau. Les ressources supervisées qui le supportent mettent à disposition une sonde qui expose leurs états et leurs configurations via une interface appelée "base d'information pour la gestion du réseau" (MIB). La solution de supervision, quant à elle, doit intégrer un composant appelé "station de gestion réseau" (NMS) qui sollicite en mode pull la ressource supervisée pour recevoir les mesures.

- **sFlow** [1] est un autre protocole qui offre des fonctionnalités de supervision. Contrairement aux sondes de SNMP, celles de sFlow poussent (push) les mesures à la solution de supervision. Initialement, ce protocole ne permet de superviser que le réseau en capturant le trafic qui y circule. Le projet Host Flow [2] permet aux ressources supervisées (par exemple, des applications, des machines virtuelles ou des machines physiques) d'intégrer des sondes qui exposent leurs états avec le protocole sFlow.

- En plus de la configuration, **OpenFlow** [ONF 2015] permet également la supervision des commutateurs des réseaux programmables (SDN). En effet, à l'aide de ce protocole, le contrôleur du réseau peut demander à un commutateur des informations sur un flux bien déterminé en envoyant un message de type "FlowStatisticsRequest". Le commutateur répond à ce message par la durée de support et le nombre d'octets du flux en question en envoyant un message de type "FlowStatisticsReply".

Pour chacun de ces protocoles nous donnons un exemple d'une solution de supervision centralisée qui s'appuie dessus :

- **Cacti** [Kundu et al. 2009] est une solution de supervision qui s'appuie sur SNMP. Cette solution est utilisée typiquement pour superviser le réseau. Elle permet notamment de visualiser des courbes représentant les mesures observées (par exemple, bande passante) via une interface web.

- **ntop** [Deri et al. 2000] est une solution de supervision qui s'appuie sur sFlow (elle représente un "collecteur" dans la terminologie de sFlow). Elle a une architecture centralisée qui assure les trois fonctions que nous avons spécifiées : "Packet capture" s'appuie sur la bibliothèque libpcap pour capturer et observer les paquets. "Packet analyser" traite les paquets et les classe dans une base de données SQL. Enfin, une interface permet à l'utilisateur d'accéder aux données collectées via un terminal ou un site web.

- **PayLess** [Chowdhury et al. 2014] est une solution de supervision qui s'appuie sur OpenFlow. Elle ne nécessite pas l'installation de sondes au niveau des commutateurs mais s'interface avec une API exposée par le contrôleur aux applications du réseau (*c.-à-d.*, API nord). Via cette interface elle accède aux fonctionnalités de OpenFlow

[1] https ://sflow.org/sFlowOverview.pdf (visité le 01/10/2018)
[2] https ://sflow.net/index.php (visité le 01/10/2018)

qui lui permet de superviser les commutateurs de l'infrastructure. Les états des ressources sont exposés via une API à d'autres applications du réseau (par exemple, applications de routage, par-feu).

Modèle hiérarchique

Les solutions qui sont conçues sur un modèle hiérarchique et qui n'ont aucune décomposition fonctionnelle reposent sur un ensemble d'éléments qui collaborent dans le traitement et l'exposition des mesures selon une hiérarchie. Nous appelons cette architecture H0, sur la figure 2.1.

dRMON [Fletcher et al. 2000] est une solution destinée à la supervision du trafic et des performances d'un réseau de communication. Il peut s'agir d'un réseau local (LAN), d'un réseau de distribution de télévision ou d'un réseau de téléphonie. Cette solution supporte SNMP et deux autres protocoles de supervision à savoir RMON et RMON2 qui lui permettent de superviser des ressources sans avoir à installer ses propres sondes. Les mesures observées sont remontées à des éléments appelés "Collectors" afin de les agréger. Chacun de ces éléments expose une API qui lui permet de s'interfacer avec l'utilisateur ou un autre élément d'un niveau supérieur selon une hiérarchie. Les éléments du plus bas niveau de la hiérarchie sont appelés "Work Group Collectors". Ils collectent des mesures sur des portions limitées du réseau. Ils ne nécessitent pas de grandes capacités en termes de CPU ou de RAM. Pour cette raison, ils peuvent être déployés dans des équipements réseau comme un commutateur ou un hub. Les élément des autres niveaux de la hiérarchie sont appelés "Domain Collectors". Ils sont utilisés pour les réseaux de grande taille (par exemple, ensemble de LAN) pour collecter les mesures remontées par les "Work Group Collectors". Ils nécessitent des capacités plus importantes que celles qui sont requises par les "Work Group Collectors".

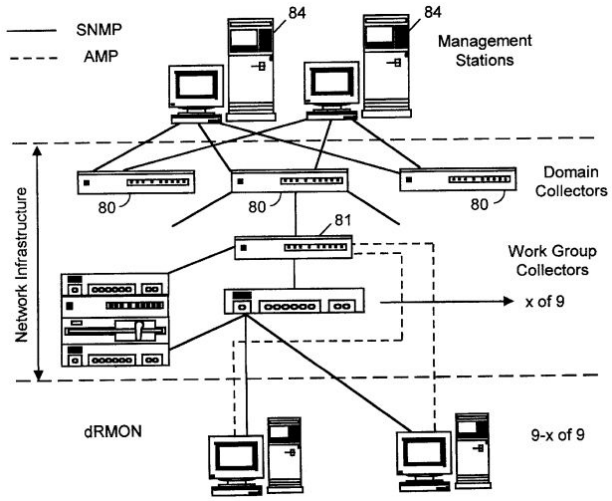

Figure 2.3 – Architecture de dRmon [Fletcher et al. 2000]
(Une hiérarchie composée des "Work Group Collectors" et des "Domain Collectors")

GEMS [Subramaniyan et al. 2006] est une solution destinée à la supervision des grilles de calcul. Comme l'illustre la figure 2.4, les éléments de cette solution s'organisent dans une hiérarchie pour augmenter la capacité globale de traitement des mesures. Chaque élément supporte nativement l'observation des métriques usuelles (par exemple, charge du CPU, utilisation de la mémoire, utilisation du réseau). Il peut s'interfacer également avec une sonde externe pour observer d'autres mesures. Il agrège les mesures observées en appliquant des requêtes SQL définies par l'utilisateur. Ensuite, il envoie les résultats à un autre élément d'un niveau supérieur L_{i+1}. Cet élément agrège à son tour les résultats calculés par tous les éléments du niveau inférieur L_i et construit ainsi une vue plus large. Pour améliorer la résilience aux pannes des serveurs, cette solution met en place un protocole de bavardage (ou gossiping [Kempe et al. 2003]). À l'aide de ce protocole, les éléments appartenant à deux niveaux successifs de la hiérarchie établissent un consensus sur la disponibilité des ressources. Cela permet de faire intervenir plusieurs éléments dans la détection des pannes et d'éviter ainsi les fausses détections qui sont dues à des pertes de messages ou à des pannes de liens. Chaque élément expose une API qui permet à l'utilisateur d'initialiser ou de configurer le protocole et d'accéder aux mesures collectées.

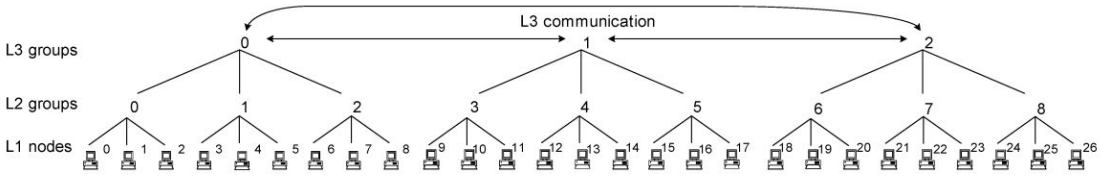

Figure 2.4 – Architecture de GEMS [Subramaniyan et al. 2006]

Modèle pair-à-pair

Les solutions qui reposent sur le modèle pair-à-pair et qui n'ont aucune décomposition fonctionnelle n'imposent pas une structure particulière de communication entre les différents éléments qui les constituent. Nous appelons cette architecture P0, sur la figure 2.1.

PeerMon [Newhall et al. 2010] est une solution destinée à la supervision du réseau. Elle s'appuie sur un ensemble de démons qui échangent les mesures selon le mode de communication pair-à-pair. Comme l'illustre la figure 2.5, chaque agent est composé de trois processus et une table de hachage qui stocke les mesures. Le premier "Client Interface" assure la fonction d'exposition. Le deuxième "Sender" assure la fonction d'observation. En plus, il remonte les mesures observées sur sa ressource hôte à trois autres démons. Réciproquement, le troisième "Listener" collecte les mesures remontées par d'autres démons. Chaque démon considère les attributs TTL des mesures observées et collectées pour vérifier qu'elles sont plus récentes que celles dont il dispose. Pour optimiser la propagation des mesures les plus récentes entre les différents démons, trois heuristiques d'envoi sont appliquées successivement. La première consiste à envoyer les mesures aux démons qui ont rejoint le réseau le plus récemment. Elle vise à intégrer de tels démons dans le système rapidement. La deuxième heuristique consiste à envoyer les

mesures aux démons "oubliés". Ce sont ceux qui ont envoyé des mesures avec les TTL les moins récents. La dernière heuristique consiste à envoyer des mesures aux démons qui ont été les moins récemment contactés. Ainsi, le risque d'isolement de démons est éliminé.

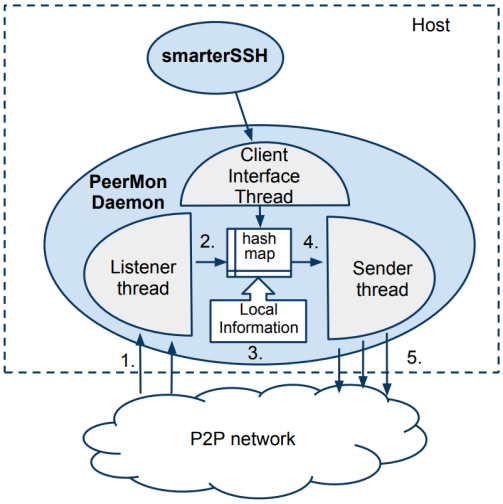

FIGURE 2.5 – Architecture de PeerMon [NEWHALL et al. 2010]
(Un pair se compose de trois processus : "Client Interface" assure la fonction d'exposition. "Sender" assure la fonction d'observation. "Listener" collecte les mesures des autres pairs)

Varanus [WARD et al. 2013] est une solution de supervision destinée aux infrastructures Cloud largement distribuées. Comme l'illustre la figure 2.6, cette solution s'appuie sur un ensemble d'agents qui observent l'état de leurs ressources hôtes et les échangent selon le modèle de communication pair-à-pair à l'aide d'un protocole de bavardage. Les agents sont répartis dans trois ensembles : "groupe", "région" et "Cloud". Le "groupe" représente un ensemble d'agents déployés sur des machines virtuelles. La "région" est constituée d'un ensemble de groupes. Elle représente un ensemble d'agents déployés au niveau d'un centre de données, une région géographique ... Le "Cloud" est constitué de l'ensemble de toutes les régions. L'objectif de cette répartition est de réduire l'empreinte de communication entre les agents tout en augmentant la vitesse de propagation des mesures. En effet, la nature et la fréquence d'échange d'informations entre les agents dépendent des ensembles auxquels ces agents appartiennent. Ceux au sein du même "groupe" échangent des mesures suite à tout changement. Les agents au sein de la même "région" mais qui appartiennent à des "groupes" différents échangent périodiquement des mesures agrégées. Les agents au sein du même "Cloud" mais qui appartiennent à des "régions" différentes échangent périodiquement des mesures agrégées à l'échelle de la région. La fréquence de ces échanges est inférieure à celle utilisée dans la communication au sein de la même "région". À noter qu'au niveau de chaque ensemble, il n'y a pas un agent désigné avec lequel doivent communiquer les agents des ensembles externe. En effet, la hiérarchie entre les ensembles n'a d'impact que sur la fréquence et la nature des mesures échangées. Pour cette raison nous classons cette solutions dans le modèle pair-à-pair et non pas dans le modèle hiérarchique.

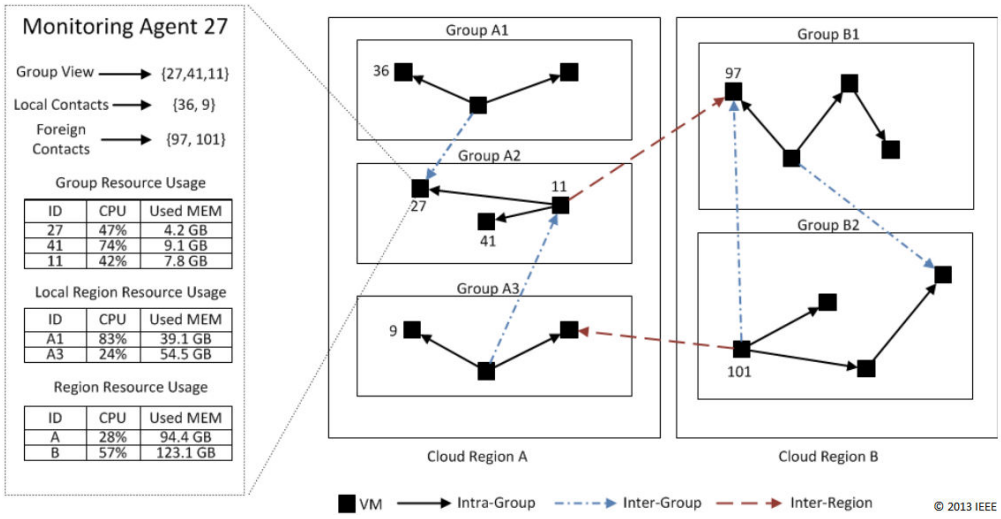

FIGURE 2.6 – Architecture de Varanus [WARD et al. 2013]
(Les agents sont répartis dans trois ensembles : "groupe", "région" et "Cloud". Chaque agent détermine les états des ensembles auxquels il appartient en collectant des mesures observées par d'autres agents)

MonALISA [LEGRAND et al. 2009] est une solution destinée à la supervision des infrastructures de Grid Computing. Son architecture est représentée sur la figure 2.7. Elle s'appuie principalement sur des agents appelés "MonALISA services". Ces derniers peuvent être déployés sur les ressources supervisées. Ils assurent des différentes fonctions : observation, filtrage, agrégation et stockage des mesures, déclenchement d'événements ou notification des utilisateurs. Ils peuvent également collecter des mesures qui sont observées par d'autres agents. Cela permet de mettre en place une coopération qui n'impose pas une hiérarchie. Les agents doivent s'identifier auprès d'autres éléments appelés "Lookup services" en déclarant des attributs qui les décrivent. Ainsi, les agents peuvent se découvrir en consultant les "Lookup services". Ces derniers maintiennent également une vue sur les états des agents. En effet, chaque agent doit leur confirmer sa présence périodiquement. Sinon, il est supprimé et une notification est envoyée aux agents qui sont intéressés par son état. Les "Lookup services" communiquent également entre eux même pour échanger des informations sur les agents. MonALISA s'appuie sur un dernier type d'éléments appelé "proxy service" qui permet de sécuriser l'accès aux agents. Il fait appel à un service d'authentification pour vérifier les droits d'accès des utilisateurs.

2.3.2 Décomposition fonctionnelle élémentaire

Les solutions ayant une décomposition fonctionnelle élémentaire s'appuient sur leurs propres sondes dans la collecte des mesures.

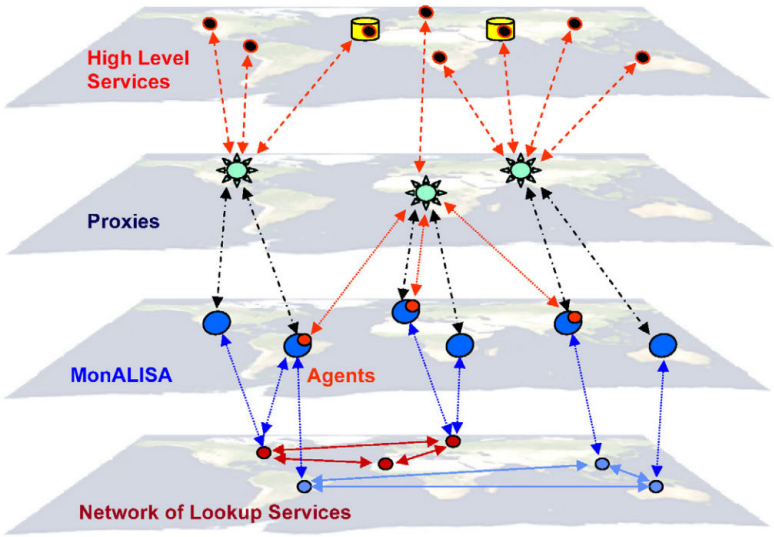

FIGURE 2.7 – Architecture de MonALISA [LEGRAND et al. 2009]

Modèle centralisé

Les solutions qui ont été conçues selon le modèle centralisé et qui ont une décomposition fonctionnelle élémentaire s'appuient sur un ensemble de sondes qui remontent les mesures à l'élément central. Ce dernier assure les fonctions de traitement et d'exposition. Nous appelons cette architecture C1, sur la figure 2.1.

Statsd [3] est une solution de supervision basique et simple à utiliser. Elle offre des sondes qui remontent les mesures à un serveur central. Les mesures sont remontées selon le mode de communication "push" afin de réduire la charge sur la ressource supervisée et avec le protocole UDP afin de réduire l'impact de perte des mesures. Le serveur central assure des fonctions de traitements basiques sur les mesures collectées comme le calcul de moyennes, des bornes maximales ou minimales... Il expose les résultats de traitement via une API qui s'intègre nativement avec l'outil de construction des tableaux de bord Graphite [4]. Les sondes et le serveur sont disponibles avec différents langages de programmation (par exemple, C, C++, Python, Ruby, Go). Cela permet d'intégrer facilement Statsd dans d'autres applications comme un composant.

Nagios [5] est une solution de supervision largement utilisée. Elle s'appuie sur deux catégories d'éléments. Les éléments de la première catégorie assurent la fonction d'observation. NCPA (Nagios Cross-Platform Agent) est leur implémentation la plus performante [NAGIOS 2014]. Les éléments de la deuxième catégorie assurent les fonctions de traitement et d'exposition. Cinq implémentations sont disponibles pour cette deuxième catégorie d'éléments : "Nagios Network Analyzer", "Nagios Fusion", "Nagios Log Server", "Nagios XI" et "Nagios Core". Nous considérons cette dernière implémentation car elle est la seule

[3] https ://github.com/etsy/statsd/wiki (visité le 01/10/2018)
[4] https ://graphiteapp.org/ (visité le 01/10/2018)
[5] https ://www.nagios.com (visité le 01/10/2018)

disponible en open source. "Nagios Core" est conçu pour fonctionner selon le modèle centralisé. Il fait appel à toutes les sondes du système pour collecter les mesures selon le modèle de communication "pull". Les détails sur les mesures à collecter et sur les traitements à faire doivent être indiqués dans un fichier de configuration. "Nagios Core" s'appuie sur ce fichier pour définir et ordonnancer les tâches à réaliser. Suite à toute modification sur les ressources supervisées, ce fichier doit être modifié et "Nagios Core" doit définir et ordonnancer de nouveau les tâches à réaliser. Ce mode de fonctionnement complique l'utilisation de Nagios pour la supervision des infrastructures dynamiques comme celles du Cloud ou du Edge.

IaaSMon [Gutierrez-Aguado et al. 2016] est une solution destinée à la supervision des infrastructures du Cloud. Elle s'appuie sur Nagios car il a une grande communauté d'utilisateurs. Pour l'adapter à la supervision des infrastructures de Cloud, IaaSMon s'appuie sur deux agents colorés en gris foncé sur la figure 2.8. Le premier est "Infrastructure Monitoring Manager" (IMM). Son rôle consiste à maintenir à jour la configuration de Nagios. Pour cela, il intercepte les communications entre les services du gestionnaire de l'infrastructure (OpenStack) et identifie les événements qui nécessitent la reconfiguration de Nagios (par exemple, le rajout de nouvelles machines physiques ou virtuelles, les allocations des adresses IP, les changements des états des machines virtuelles, le changement de la topologie du réseau). Il définit par la suite les reconfigurations nécessaires et les transfère à NConf [6] qui les applique sur Nagios. Le deuxième agent est "Cloud Monitoring Checks" (CMC). Il permet l'accès aux machines virtuelles afin de les superviser. Cela nécessite d'identifier leurs VLANs. Ces derniers sont mis en place en utilisant des names spaces différents au niveau de la machine physique. Le rôle du CMC consiste à remonter un inventaire de ces names spaces pour permettre l'accès aux machines virtuelles.

Modèle hiérarchique

Les solutions qui ont été conçues selon le modèle hiérarchique et qui ont une décomposition fonctionnelle élémentaire s'appuient sur un ensemble d'éléments qui coopèrent selon une hiérarchie pour le traitement des mesures observées par les sondes. Nous appelons cette architecture H1, sur la figure 2.1.

Ganglia [Massie et al. 2004] est une solution destinée à la supervision des grilles de calcul. Son architecture est représentée sur la figure 2.9. Elle se compose de deux types d'éléments : les "Ganglia monitoring daemon" (gmond) qui assurent la fonction d'observation et les "Ganglia meta daemon" (gmetad) qui assurent les fonctions de traitement et d'exposition. Ces éléments sont structurés selon une arborescence. Les feuilles de l'arbre sont les gmonds qui observent les ressources. Ils remontent les mesures à des gmetads qui sont dédiés pour chaque grappe de serveurs. Ces gmetads remontent à leur tour les résultats de leurs traitements à d'autres gmetads qui agrègent les états de plusieurs grappes de serveurs. Chaque gmetad de l'arbre dispose d'un fichier où ses sources de mesures (gmonds ou gmetads) sont identifiées par leurs adresses IP. Il les interroge périodiquement pour obtenir les mesures en tant que fichier XML. Il analyse

[6] http ://www.nconf.org/ (visité le 01/10/2018)

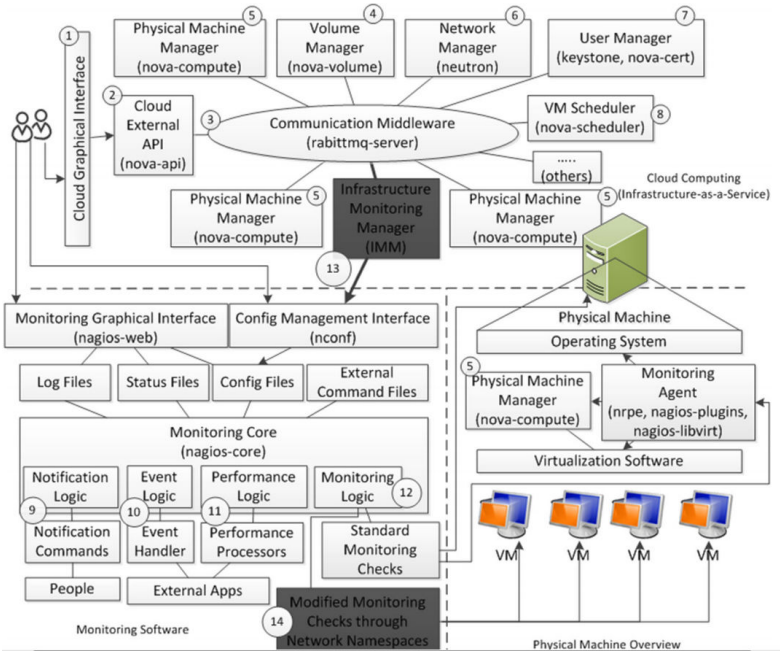

FIGURE 2.8 – Architecture de IaaSMon [GUTIERREZ-AGUADO et al. 2016]
(IMM (13) intercepte les communications entre les services d'Openstack et CMC (13) intercepte les names spaces utilisés au niveau des machines physiques)

le fichier et stocke les mesures en tant que séries temporelles dans une base de données Round Robin qui les agrège pour conserver une taille constante. Enfin, ces mesures sont exposées à l'utilisateur ou au gmetad du niveau supérieur.

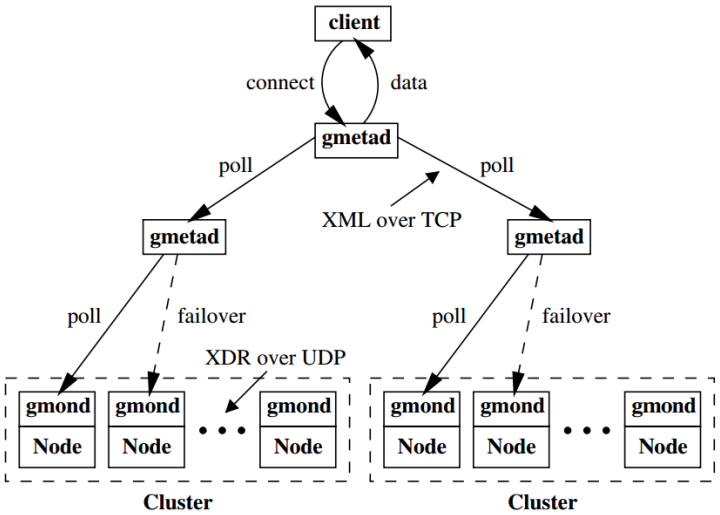

FIGURE 2.9 – Architecture de Ganglia [MASSIE et al. 2004]

Prometheus [7] est une solution destinée à la supervision des ressources dynamiques basée sur l'architecture micro-service ou sur le modèle de Cloud Computing. Son architecture est représentée sur la figure 2.10. Pour s'adapter à la dynamité des ressources supervisées, cette solution s'appuie sur un ensemble d'éléments autonomes appelés "serveurs Promoteus" qui peuvent collaborer. En effet, chacun de ces éléments peut traiter des mesures remontées par un autre "serveur Promoteus". Ainsi, une hiérarchie de serveurs peut être mise en place et adaptée en fonction de la charge de supervision. Promoteus s'appuie sur des sondes appelées "Exporters" qui remontent les mesures en mode "pull". Pour les ressources qui exposent leurs mesures en mode "push", l'élément "pushgateway" a été conçu afin de transformer leur mode de communication en "pull". Chaque "serveur Promoteus" stocke ses mesures localement et non pas dans un cluster de stockage. Cela permet de distribuer le stockage et d'améliorer ainsi la résilience aux pannes. Les résultats de traitement de chaque "serveur Promoteus" sont accessibles via une interface web, une API ou l'outil de visualisation Grafana [8]. Les alarmes peuvent être consommées par "alarm manager" qui se charge de notifier les utilisateurs par courriel ou via PagerDuty par exemple.

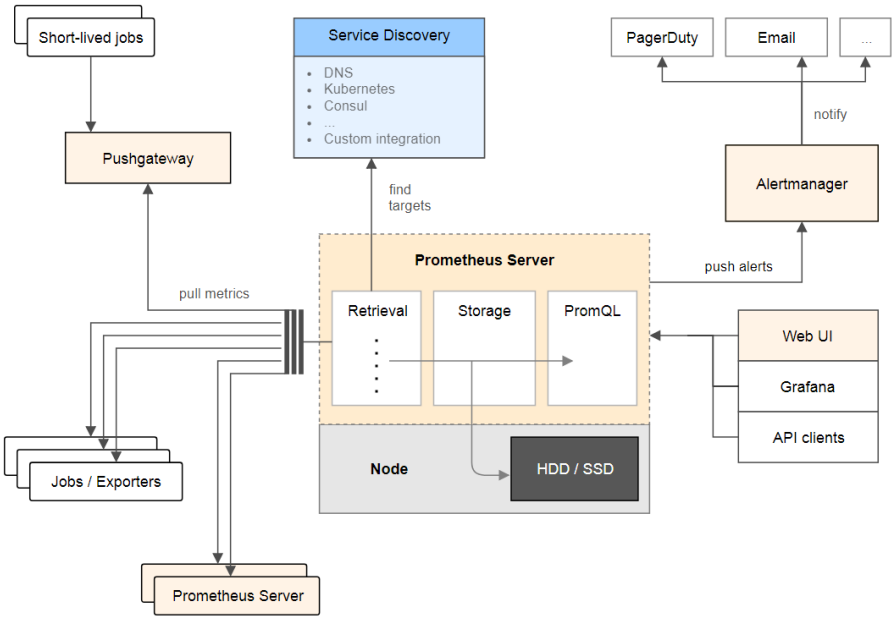

FIGURE 2.10 – Architecture de Prometheus [9]

Modèle pair-à-pair

Les solutions qui ont été conçues selon le modèle pair-à-pair et qui ont une décomposition fonctionnelle élémentaire n'imposent pas une structure particulière sur les éléments

[7] https ://prometheus.io/ (visité le 01/10/2018)
[8] https ://grafana.com/ (visité le 01/10/2018)

qui assurent le traitement des mesures observées par les sondes. Nous appelons cette architecture P1, sur la figure 2.1.

[Seok et al. 2010] proposent une solution pour la supervision des réseaux de capteurs. Elle s'appuie essentiellement sur deux types d'éléments. Le premier "Sink" assure la fonction d'observation. Il fait appel aux capteurs auxquels il est connecté en ZigBee. Il peut également récupérer les mesures collectées par d'autres "Sink". Le deuxième "Peer" assure les fonctions de traitement. Il peut communiquer à tout autre élément du système de supervision selon le modèle de communication pair-à-pair. Un réseau "overlay network" est mis en place pour remonter les mesures collectées par les "Sink" aux "Peer". Ce réseau s'appuie sur le système de routage Pastry [Rowstron et al. 2001] destiné pour les réseaux pair-à-pair. Une interface web est mise à la disposition des utilisateurs qui ne désirent pas déployer des "Peer" propres. Elle se charge de les orienter vers des "Peer" déployés au préalable.

2.3.3 Décomposition fonctionnelle avancée

Les solutions ayant une décomposition fonctionnelle avancée s'appuient sur des fonctions élémentaires distribuées.

Modèle centralisé

Les solutions qui ont été conçues selon le modèle centralisé et qui ont une décomposition fonctionnelle avancée s'appuient sur des éléments qui réalisent des traitements élémentaires et qui communiquent tous avec un élément centralisé. Nous appelons cette architecture C*, sur la figure 2.1.

Storm [Toshniwal et al. 2014] est une solution destinée au traitement des flux de données. Cette solution est utilisée également pour la supervision puisque le flux de mesures est un flux de données. Son architecture est représentée sur la figure 2.11. Son fonctionnement consiste à exécuter une "topologie" qui est une description de l'ensemble des traitements à réaliser. Elle est exprimée sous forme d'un graphe orienté. Ses nœuds peuvent avoir deux types : des "spots" qui sont les sources de flux (par exemple, sondes) ou des "bolts" qui réalisent des traitements élémentaires (par exemple, filtrage, jointure, agrégation). Ses arcs sont les flux de mesures. La topologie est soumise à un serveur central appelé "Nimbus". Il est responsable de la distribution et la coordination de l'exécution de la topologie. Il lance et configure des processus "worker" sur des machines distantes afin d'exécuter des portions de la topologie. "Nimbus" gère les "worker" exécutés au niveau de la même machine par le même élément appelé "supervisor" déployé localement. Le "supervisor" reporte périodiquement leurs états au "Nimbus". S'ils tombent en panne, il les redémarre. Il reporte aussi au "Nimbus" la capacité du nœud hôte à exécuter d'autres topologies. Toutes les communications entre "Nimbus" et les éléments de type "supervisor" passent par un serveur centralisé ZooKeeper [10]. Ce dernier enregistre leurs états. Ainsi, si l'un d'eux tombe en panne, il peut restituer son état en se connectant à ZooZeeper.

[10] https ://zookeeper.apache.org/ (visité le 01/10/2018)

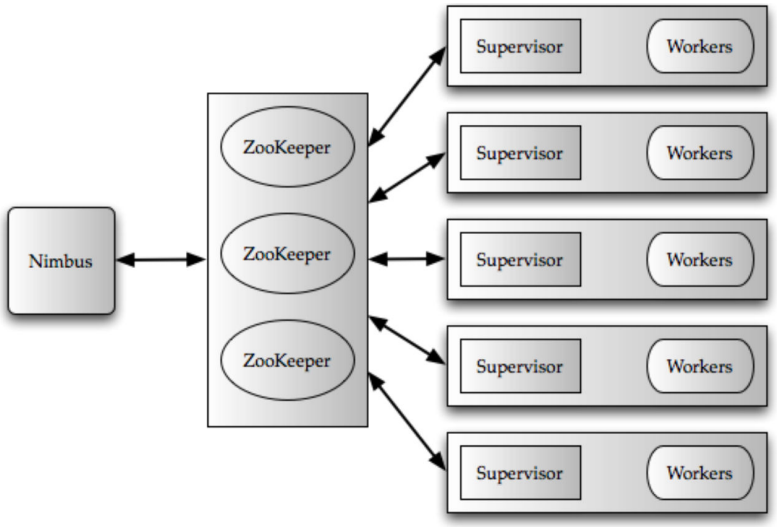

FIGURE 2.11 – Architecture de Storm [TOSHNIWAL et al. 2014]

Monasca [11] est une solution qui offre la supervision en tant que service utilitaire (Monitoring as a Service) aux tenants d'une infrastructure Cloud. Son architecture est représentée sur la figure 2.12. Son élément "Monasca REST API" assure la fonction d'exposition. Il met à la disposition des utilisateurs une API REST qui est accessible via l'interface en ligne de commande "Monasca CLI", l'interface graphique Horizon [12] d'OpenStack ou l'outil de visualisation Grafana [13]. Cette interface permet aux utilisateurs de configurer les fonctionnalités attendues (par exemple, traitement des mesures, déclenchement d'alarmes) ainsi que l'accès aux mesures collectées. Elle est utilisée également par les éléments "Monasca-agent" qui assurent la fonction d'observation pour remonter les mesures en mode "push". Le traitement de mesures est assuré par la fonction "Threshold Engine". Cet élément s'appuie essentiellement sur la solution de traitement des flux de données Storm [TOSHNIWAL et al. 2014]. Il déclenche des événements qui sont traités par l'élément "Notification Engine". Le stockage de mesures et des alarmes est assuré par l'élément "Persister" qui s'appuie sur la base de données de séries temporelles Influx [14]. Les communications entre les éléments "Monasca REST API", "Threshold Engine", "Notification Engine" et "Persister" passent toutes via Kafka [15]. C'est un bus de communication qui permet le stockage des messages de manière distribuée. Cependant, il s'appuie sur la solution centralisée ZooKeeper [16] pour la synchronisation entre les différentes portions du bus. C'est pour cette raison que nous le classons dans le modèle centralisé.

[11] http ://monasca.io (visité le 01/10/2018)
[12] https ://wiki.openstack.org/wiki/Horizon (visité le 01/10/2018)
[13] https ://grafana.com/ (visité le 01/10/2018)
[14] https ://www.influxdata.com/ (visité le 01/10/2018)
[15] https ://kafka.apache.org/ (visité le 01/10/2018)
[16] https ://zookeeper.apache.org/ (visité le 01/10/2018)

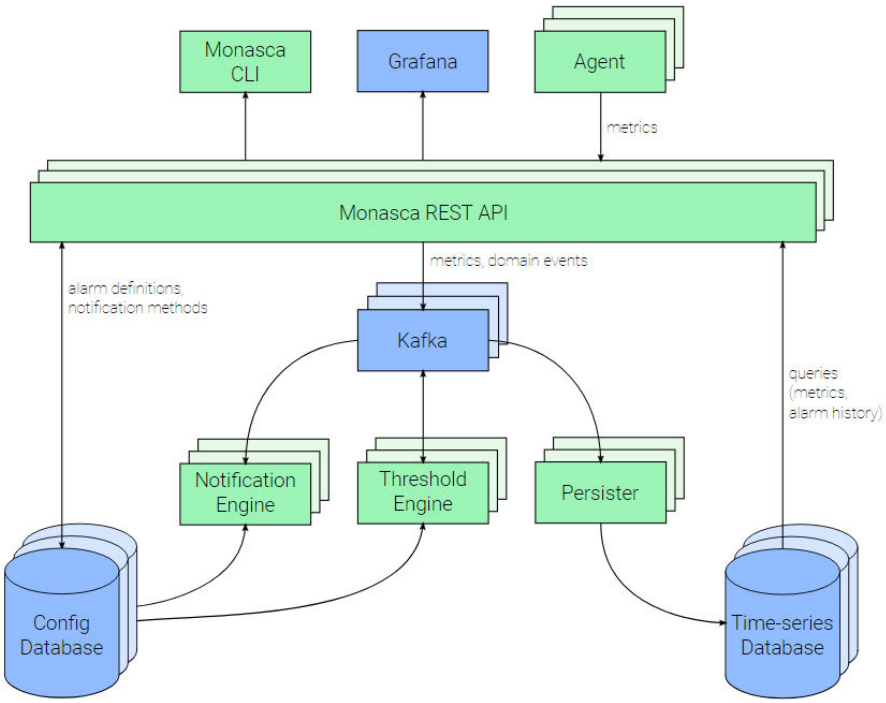

FIGURE 2.12 – Architecture de Monasca [17]

Modèle hiérarchique

Les solutions qui ont été conçues selon le modèle hiérarchique et qui ont une décomposition fonctionnelle avancée s'appuient sur des éléments qui réalisent des traitements élémentaires. Ils sont organisés selon deux structures. La première les organise selon un chaînage fonctionnel. La deuxième structure organise les éléments qui réalisent la même fonction du chaînage selon une hiérarchie. Nous appelons cette architecture H*, sur la figure 2.1.

Hierarchical-CEP [XU et al. 2014b] est une solution destinée pour la supervision des réseaux de capteurs. Son architecture est représentée sur la figure 2.13. Elle s'appuie sur un ensemble de processeurs d'événements complexes afin de traiter les mesures. Ils réalisent une série d'opérations logiques (ou, et, non) ou temporelles (par exemple, vérifier que certains événements se déroulent selon un ordre spécifique). Ils sont répartis hiérarchiquement sur trois types de nœuds : "sensor", "gateway" et "server". Au niveau des nœuds de type "sensor", ils réalisent un traitement préliminaire sur les mesures collectées par les capteurs. Ils remontent les résultats de traitement à d'autres processeurs au niveau des nœuds de type "gateway". Ces processeurs réalisent un traitement plus avancé grâce aux ressources plus performantes dont ils disposent. Ils remontent à leur tour les résultats de traitement à d'autres processeurs au niveau des nœuds de type "server" qui constituent le plus haut niveau de l'hiérarchie. Ces processeurs construisent une vue globale du système supervisé et l'exposent aux utilisateurs. Ce sont ces processeurs qui sont responsables également de la répartition du traitement dans l'hiérarchie. Ils reçoivent des requêtes de supervision des utilisateurs qu'ils modélisent par un automate

fini non déterministe (*c.-à-d.*, un graphe dont les nœuds représentent des états et dont les arcs représentent les changements d'état). Cet automate peut contenir des cycles. Pour cela, un algorithme est utilisé pour les identifier et les transformer en des nœuds non décomposables. Cette étape permet de transformer la requête en un graphe acyclique. Ce dernier est ensuite réparti sur la hiérarchie de processeurs par un autre algorithme qui prend en considération les capacités des ressources hôtes.

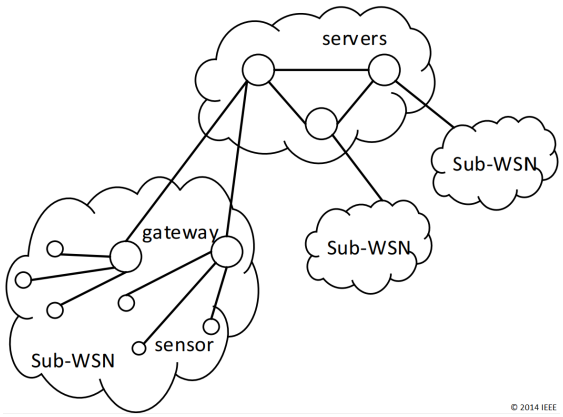

FIGURE 2.13 – Architecture de Hierarchical-CEP [XU et al. 2014b]

Modèle pair-à-pair

Dans cette section, nous considérons les solutions qui reposent sur le modèle pair-à-pair et qui ont une décomposition fonctionnelle avancée. Elles s'appuient sur des éléments qui réalisent des traitements élémentaires et qui communiquent pour assurer un chaînage fonctionnel ou pour partager des informations sur le système. Nous appelons cette architecture P*, sur la figure 2.1.

À notre connaissance, il n'y a pas de solutions de supervision qui respectent cette architecture.

2.3.4 Modèles hybrides

La majorité des solutions de supervision existantes ont été conçues selon les trois modèles architecturaux : centralisé, hiérarchique ou pair-à-pair. Cependant, certaines d'entre elles appliquent des modèles hybrides. Cela reste insuffisant pour vérifier les propriétés qui ne sont pas satisfaites par l'un de leurs modèles même s'il permet de mieux y convenir.

Modèle centralisé et hiérarchique

Morin et al. proposent une solution de supervision [MORIN et al. 2009] qui combine les deux modèles, centralisé et hiérarchique, et qui a une décomposition fonctionnelle avancée. La fonction d'observation et une partie de la fonction du traitement est assuré par WildCat [DAVID et al. 2005]. Ce dernier est un cadre de travail générique qui permet

de construire des applications conscientes de leurs environnements. Il s'appuie sur un ensemble d'éléments qui assurent la fonction d'observation et de traitement. Comme l'illustre la figure 2.14, ils sont structurés selon une arborescence appelée "context". Les nœuds de l'arbre peuvent avoir l'un de deux types. Le premier est "attribut". Il représente une métrique observée (par exemple, Mémoire disponible, charge CPU, bande passante disponible). Le deuxième type de nœuds est "resource". Il représente l'élément concerné par la métrique (par exemple, mémoire, CPU, composant logiciel) ou un élément par lequel doivent transiter les métriques. Un traitement poussé des mesures est réalisé à la racine de l'arbre par Esper [Bernhardt et al. 2007]. C'est un élément centralisé qui est destiné au traitement d'événements complexes. C'est cet élément qui assure également la fonction d'exposition.

En exécutant une partie du traitement selon une arborescence grâce à WildCat, cette solution améliore la capacité de *passage à l'échelle* d'Esper. Cependant, cela reste insuffisant pour satisfaire cette propriété car la grande partie du traitement est centralisée. En plus, cette architecture ne permet pas de satisfaire la *proximité vis-à-vis des utilisateurs* puisque le traitement de mesures ne peut pas être réalisé entièrement à proximité des utilisateurs mais doit passer par un élément central.

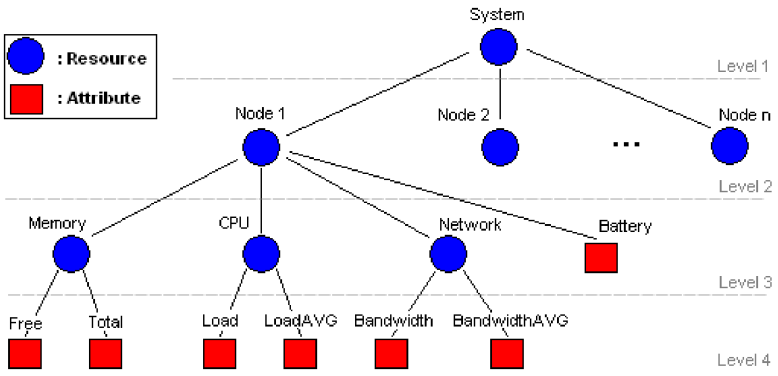

Figure 2.14 – Exemple d'instanciation de WildCat [Morin et al. 2009]

Modèle centralisé et pair-à-pair

visPerf [Lee et al. 2003] est une solution destinée à la supervision des grilles de calcul. Elle combine les deux modèles, centralisé et pair-à-pair, et a une décomposition fonctionnelle avancée. Elle s'appuie sur un ensemble d'éléments "visSensor" qui assurent la fonction d'observation. Ils observent les ressources en utilisant des outils de supervision d'UNIX (par exemple, vmstat, iostat, top) ou en analysant les fichiers log. Les ressources sont partitionnées en des ensembles de grappes de serveurs appelées "Local Grid". Au niveau, de chaque ensemble il y a un "visSensor" appelé "Local representative monitor" qui construit des mesures globales sur l'état du cluster. Les autres "visSensor" appelées "Local monitoring entity" collectent des mesures plus détaillées sur leurs serveurs hôtes. Les "Local representative monitor" échangent l'état du système supervisé selon le modèle de communication pair-à-pair comme l'illustre la figure 2.15. Un élément centralisé "visPerf

controller" traite les mesures observées par les "visSensor" et expose l'état des ressources aux utilisateurs. Il permet de configurer les différents éléments du système de supervision (par exemple, changer la nature des mesures collectées ou changer la fréquence de collecte).

Le "visPerf controller" représente un point de défaillance critique. En plus, il ne permet pas de passer à l'échelle ni d'exécuter le traitement à proximité des utilisateurs. Par conséquent, les propriétés : *passage à l'échelle*, *résilience aux pannes/volatilité des serveurs* et *proximité vis-à-vis des utilisateurs* ne sont pas satisfaites.

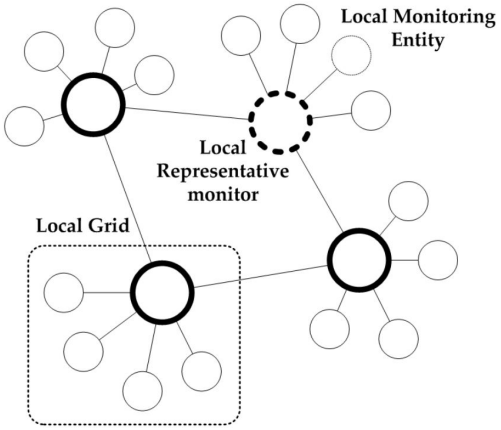

FIGURE 2.15 – Architecture de visPerf [LEE et al. 2003]

Modèle hiérarchique et pair-à-pair

[SALVADOR et al. 2005] proposent une solution pour la supervision du réseau. Elle combine les deux modèles, hiérarchique et pair-à-pair ; elle n'a pas de décomposition fonctionnelle. Comme l'illustre la figure 2.16, cette solution s'appuie sur des éléments monolithiques répartis sur différentes portions du réseau. Ces éléments peuvent avoir l'un des deux rôles : "Probe" ou "Super-probe". Chaque "Super-probe" est responsable d'un ensemble de "Probes" qui lui remontent les mesures selon le mode de communication maître-esclave. Le "Super-probe" agrège les mesures collectées et les communique aux autres "Super-probe". Il n'y a pas d'hiérarchie, par contre, dans la communication entre les "Super-probe", chacun d'eux communique avec les autres selon le mode de communication pair-à-pair. Par défaut, il y a un seul "Super-probe" à chaque portion du réseau. Il est le maître des "Probes" qui supervisent cette portion du réseau. Cependant, il est possible de s'appuyer sur plusieurs "Super-probe" au niveau du même groupe pour passer à l'échelle. En effet, chaque "Super-probe" compare son taux d'utilisation moyen de CPU, de RAM, de stockage, bande passante aux capacités disponibles. S'il n'y a pas assez de ressources, il essaie d'associer ses "Probe" aux "Super-probe" des autres groupes. S'il n'y a pas d'éléments qui disposent des ressources nécessaires, il change le rôle de l'un de ses "Probe" en "Super-probe" et lui associe certains "Probe". Un "Super-probe" peut également changer son propre rôle en "Probe" s'il n'a pas une charge considérable et s'il constate que d'autres "Super-probe" en disposent.

Cette architecture permet à la solution de satisfaire le *passage à l'échelle* et la *proximité vis-à-vis des utilisateurs*. Cependant, si le serveur hôte d'un "Super-probe" tombe en panne tous les "Probes" qui lui sont associés ne seront plus joignables. Ainsi, la *tolérance aux pannes/volatilité des serveurs* n'est pas satisfaite.

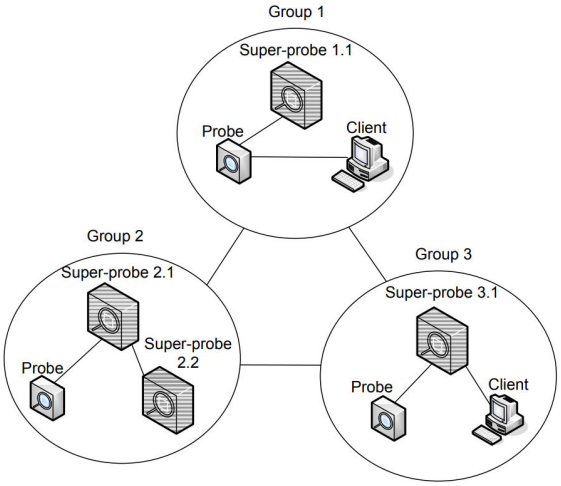

FIGURE 2.16 – Architecture de [SALVADOR et al. 2005]

2.4 Synthèse

La table 2.3 illustre l'évaluation des solutions de supervision en fonction de leurs architectures. L'architecture qui satisfait au mieux les propriétés spécifiées est P*. Sa décomposition fonctionnelle avancée permet de satisfaire la *compatibilité vis-à-vis des ressources* et la *résilience aux changements/pannes de réseau*. En plus, son modèle pair-à-pair n'impose pas une hiérarchie. Cela permet de satisfaire la *Résilience aux apparitions/disparition des serveurs*, la *proximité vis-à-vis des utilisateurs* et le *passage à l'échelle*. Aujourd'hui, aucune solution répondant à ce modèle n'existe. La difficulté de sa conception est en partie due à la complexité de son architecture de déploiement (placement initial des fonctions et maintenance de l'architecture en fonction des besoins).

La stratégie de déploiement et de maintenance de l'architecture de supervision peut être réalisée selon deux approches différentes. Dans la première, les pairs qui traitent les mesures observées sont complètement autonomes. Ils maintiennent eux-mêmes leur architecture de déploiement : chacun d'eux identifie ceux avec lesquels il doit établir une communication et se positionne en fonction des besoins et de l'infrastructure sous-jacente. C'est la configuration adoptée par **PeerMon** [NEWHALL et al. 2010], **Varanus** [WARD et al. 2013] et la solution proposée par Seok et al. [SEOK et al. 2010]. Dans la deuxième configuration, des éléments dédiés maintiennent l'architecture de déploiement. Ils assurent la découverte et la coordination entre les pairs. C'est par exemple l'approche choisie par **MonALISA** [LEGRAND et al. 2009].

Bien que la première approche permet de s'abstraire d'une vue centralisée pour le

contrôle des fonctions de supervision, les stratégies de déploiement qui en découlent sont coûteuses en termes d'échange de messages et souvent peu efficaces car elles ne permettent pas d'optimiser globalement le système. Nous avons choisi la deuxième approche car elle est plus adaptée au contexte de notre étude qui est fortement dynamique (cela se manifeste à travers les changements des besoins des utilisateurs et les changements de la configuration de l'infrastructure). Concrètement, nous proposons la mise en place d'un gestionnaire construit autour d'une boucle de contrôle autonomique MAPE [Computing 2003]. Il est en charge de gérer le placement et le déploiement des fonctions de supervision durant leur cycle de vie.

2.5 Conclusion

En considérant les spécificités de l'infrastructure Edge, nous avons défini les propriétés que doit assurer une solution destinée à sa supervision. Ces propriétés sont : la *compatibilité vis-à-vis des ressources*, le *passage à l'échelle*, la *proximité vis-à-vis des utilisateurs*, la *résilience aux pannes/volatilité des serveurs*, la *résilience aux changements/pannes du réseau*. Étant donné le grand nombre de solutions de supervision existantes et la difficulté de concevoir tous les scénarios possibles pour leur évaluation, nous avons opté pour une approche qualitative. Elle consiste à considérer deux aspects dans les architectures des solutions existantes : le modèle architectural et la décomposition fonctionnelle car ils répondent différemment aux propriétés spécifiées. Nous avons identifié essentiellement trois niveaux de décompositions fonctionnelles : aucune, élémentaire et avancée ainsi que trois modèles architecturaux : centralisé, hiérarchique et pair-à-pair. L'architecture qui répond au mieux aux propriétés spécifiées est celle qui a une décomposition fonctionnelle avancée et un modèle pair-à-pair. Cependant, la mise en œuvre de pairs complètement autonomes est particulièrement complexe dans un environnement dynamique comme celui que nous adressons. En conséquence, nous avons opté pour la mise en place d'une boucle de contrôle autonomique MAPE qui gère les fonctions de supervision. Dans le chapitre suivant, nous présentons le canevas logiciel que nous avons conçu à cette fin.

Architecture	Solution de supervision	Satisfaction des propriétés requises				
		Proximité vis-à-vis des utilisateurs	Passage à l'échelle	Résilience aux pannes des serveurs	Résilience aux pannes du réseau	Compatibilité vis-à-vis des ressources
C0	NFM [Suneja et al. 2014] Cacti [Kundu et al. 2009] ntop [Deri et al. 2000] PayLess [Chowdhury et al. 2014]	-	-	-	-	-
H0	dRMON [Fletcher et al. 2000] GEMS [Subramaniyan et al. 2006]	+	+	-	-	-
P0	PeerMon [Newhall et al. 2010] Varanus [Ward et al. 2013] MonALISA [Legrand et al. 2009]	+	+	+	-	-
C1	Statsd [a] Nagios [b] IaaSMon [Gutierrez-Aguado et al. 2016]	-	-	-	+	-
H1	Ganglia [Massie et al. 2004] Prometheus [c]	+	+	-	+	-
P1	[Seok et al. 2010]	+	+	+	+	-
C*	Storm [Toshniwal et al. 2014] Monasca [d]	-	-	-	+	+
H*	Hierarchical-CEP [Xu et al. 2014b]	+	+	-	+	+
P*	-	+	+	+	+	+
CH*	[Morin et al. 2009]	-	-	-	+	+
CP*	visPerf [Lee et al. 2003]	-	-	-	+	+
HP0	[Salvador et al. 2005]	+	+	-	-	-

Table 2.3 – Évaluation des solutions de supervision selon leurs architectures

[a] https ://github.com/etsy/statsd/wiki (visité le 01/10/2018)
[b] https ://www.nagios.com (visité le 01/10/2018)
[c] https ://prometheus.io/ (visité le 01/10/2018)
[d] http ://monasca.io (visité le 01/10/2018)

Contribution

3

Canevas logiciel pour la supervision d'une infrastructure Edge

Sommaire

Dans ce chapitre, nous présentons le canevas logiciel que nous avons conçu. Il a comme objectif la mise en place d'un service de supervision qui s'appuie sur l'architecture P* décrite précédemment et qui est géré par une boucle de contrôle autonomique MAPE.

Nous commençons par exposer les différents éléments du canevas et leurs rôles dans l'architecture du service de supervision. Ensuite, nous nous concentrons spécifiquement sur trois éléments parmi eux. Ce sont le langage de description des besoins des utilisateurs, le langage de description de l'infrastructure et le calculateur de placement. Nous les présentons et nous les illustrons par un cas d'utilisation.

3.1 Architecture du service de supervision

Comme nous l'avons détaillé dans le chapitre précédent, l'architecture qui convient au mieux à notre besoin est l'architecture P*. Pour automatiser sa reconfiguration, nous proposons de nous appuyer sur une boucle de contrôle autonomique MAPE.

La boucle MAPE doit reconfigurer le service de supervision pour satisfaire les besoins fonctionnels et de qualité de service des utilisateurs. Ces besoins peuvent être très diversifiés car comme détaillé dans la section 1.4, le service de supervision est destiné aux différents tenants de l'infrastructure Edge (*c.-à-d.*, opérateur, fournisseur d'une ressource Edge, fournisseur de services et utilisateur de services) qui ont des besoins variés. Pour faire face à ce défi, nous définissons un **langage qui unifie la description des besoins de supervision des utilisateurs**. Il est présenté dans la section 3.2.

Pour vérifier la satisfaction des besoins qui concernent la qualité de service, une description des caractéristiques des ressources de l'infrastructure est requise. Cependant, comme détaillé dans la section 1.3.3, les ressources de l'infrastructure Edge (par exemple, les machines physiques/virtuelles, les routeurs, les passerelles domestiques, les liens filaires et sans fil) ont des fonctionnalités et des capacités d'hébergement différentes. Pour faire face à ce défi, nous définissons, dans la section 3.3, **un langage qui unifie la description de l'infrastructure**.

La fonction "analyse" de la boucle MAPE est assurée par le module **calculateur de placement**. En s'appuyant sur une description des besoins des utilisateurs et une description de l'infrastructure qu'il reçoit en entrée, il détermine les besoins qui ne sont pas satisfaits.

Le **calculateur de placement** assure également une partie de la fonction "planification" de la boucle MAPE. C'est cet élément qui est chargé de déterminer l'architecture de déploiement des fonctions de supervision. Pour vérifier que cette architecture satisfait les besoins de qualité de service des utilisateurs, il calcule son placement sur l'infrastructure en considérant les caractéristiques des ressources hôtes. Nous présentons les principes sur lesquels nous nous sommes appuyés pour la conception du **calculateur de placement** dans la section 3.4.

La deuxième partie de la fonction "planification" est assurée par l'**ordonnanceur de placement**. Cet élément reçoit en entrée l'architecture de déploiement identifiée par le **calculateur de placement**. Il détermine et ordonnance les tâches nécessaires pour déployer cette architecture sur l'infrastructure.

L'**ordonnanceur de placement** assure également la fonction "exécution" de la boucle MAPE. Selon l'ordonnancement établi, il reconfigure et reprogramme les fonctions de supervision déployées ou déploie de nouvelles fonctions. Comme nous l'aborderons

lors de la conclusion, la conception de l'**ordonnanceur de placement** est un vrai défi en soi. Nous avons débuté des travaux préliminaires notamment au travers d'un stage afin de réaliser le socle technologique permettant un déploiement réel des fonctions de supervision.

La fonction "monitoring" de la boucle MAPE est assurée par des fonctions de supervision qui sont déployées préalablement pour répondre aux besoins de gestion du service de supervision. Elles remontent au **calculateur de placement** une description de l'infrastructure qui est exprimée à l'aide du **langage de description**.

Ainsi, le canevas logiciel s'appuie sur deux langages de description (*c.-à-d.*, **langage de description des besoins des utilisateurs** et **langage de description de l'infrastructure**) et deux modules (*c.-à-d.*, **calculateur de placement** et **ordonnanceur de placement**) qui assurent la gestion des fonctions de supervision. L'articulation de ces éléments dans l'architecture du service de supervision est représentée par la figure 3.1.

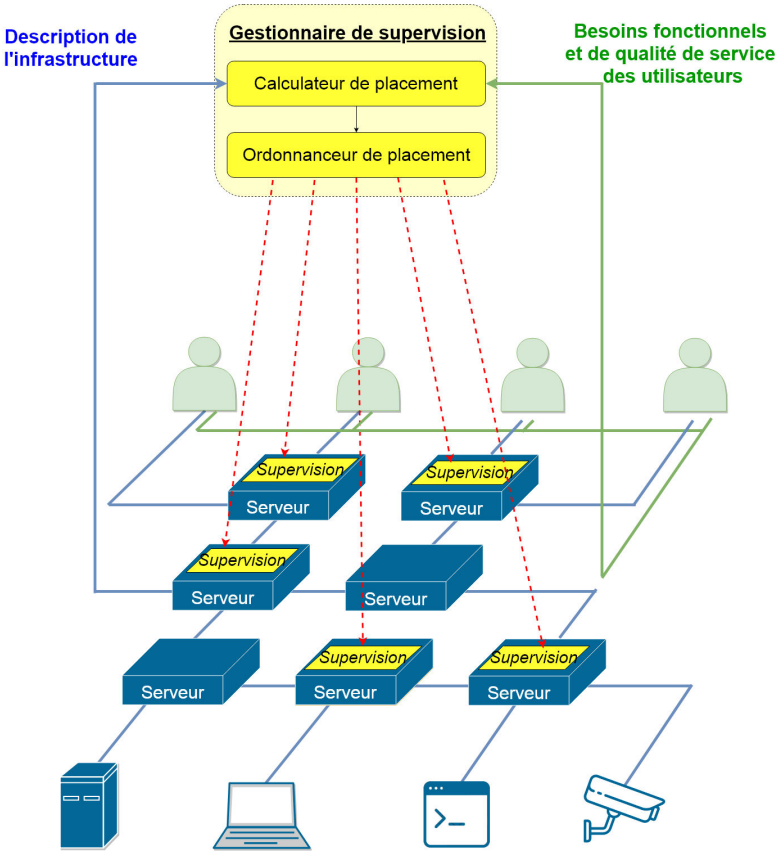

FIGURE 3.1 – Architecture du service de supervision

3.2 Langage de description des besoins des utilisateurs

Les solutions monitoring-as-a-service existantes (par exemple, Monasca [1], MISURE [Smit et al. 2013]) s'appuient sur des solutions de traitement de flux (par exemple, Apache Storm [Toshniwal et al. 2014]) et s'appuient, par conséquent, sur les langages qu'elles offrent (par exemple, summingbird [Boykin et al. 2014] de Storm) pour exprimer les requêtes de traitement des utilisateurs. Cependant, ces langages sont destinés à des usages généraux. Par conséquent, ils permettent d'exprimer des besoins de supervision des utilisateurs de différentes manières. Cela risque de rendre complexe la programmation des fonctions de supervision par l'**ordonnanceur de placement**. Pour cette raison, nous définissons un nouveau langage spécifique pour l'expression des besoins de supervision.

Le langage que nous proposons est illustré par un diagramme de classes dans la figure 3.2. Chaque classe représente une entité du lexique. Les attributs de la classe sont des caractéristiques à spécifier pour chaque entité. Les associations entre les classes sont définies selon les règles du langage. Un exemple d'utilisation de ce langage est illustré dans la section 3.5.

3.2.1 Lexique

Dans le chapitre 2, nous avons analysé les solutions de supervision existantes et identifié les fonctionnalités de base (*c.-à-d.*, observation, traitement, exposition) qu'elles peuvent offrir à leurs utilisateurs. En décomposant chaque fonctionnalité en des fonctions élémentaires, nous définissons les entités sur lesquelles s'appuie le lexique de notre langage. Nous modélisons chaque flux de mesures fl échangé entre les entités par un ensemble infini de n-uplets : $fl = \{ u \mid u \in \mathbb{R}^n \}$ car chaque élément du flux peut être un ensemble de valeurs (par exemple, mesure et instant d'observation).

La première entité est **sonde**. C'est une fonction d'observation élémentaire. Elle génère un flux constitué typiquement par des couples $(t; m)$ tel que m est une mesure et t est l'instant de son observation. La **sonde** se caractérise par la *ressource* observée et la *métrique* observée au niveau de la ressource (par exemple, RAM, CPU, bande passante).

Nous définissons cinq entités qui constituent des fonctions de traitement élémentaires :

- **Agréger :** Cette entité génère un flux qui résulte de l'application d'une fonction mathématique (par exemple, moyenne, variance, écart type) sur les éléments du flux entrant qui appartiennent à la même fenêtre temporelle (*c.-à-d.*, intervalle de temps). Elle se caractérise ainsi, par la *fonction* mathématique à appliquer et par la *fenêtre* temporelle.

- **Rajouter :** Cette entité modifie le n-uplet en lui rajoutant un élément supplémentaire généré par une *fonction* qui la caractérise. En plus de cette fonction, l'entité **rajouter** se caractérise par l'*indice* dans le n-uplet où l'élément généré doit être inséré. Un exemple d'utilisation possible de cette fonction est le marquage de l'instant de passage de chaque n-uplet.

[1] http ://monasca.io (visité le 01/10/2018)

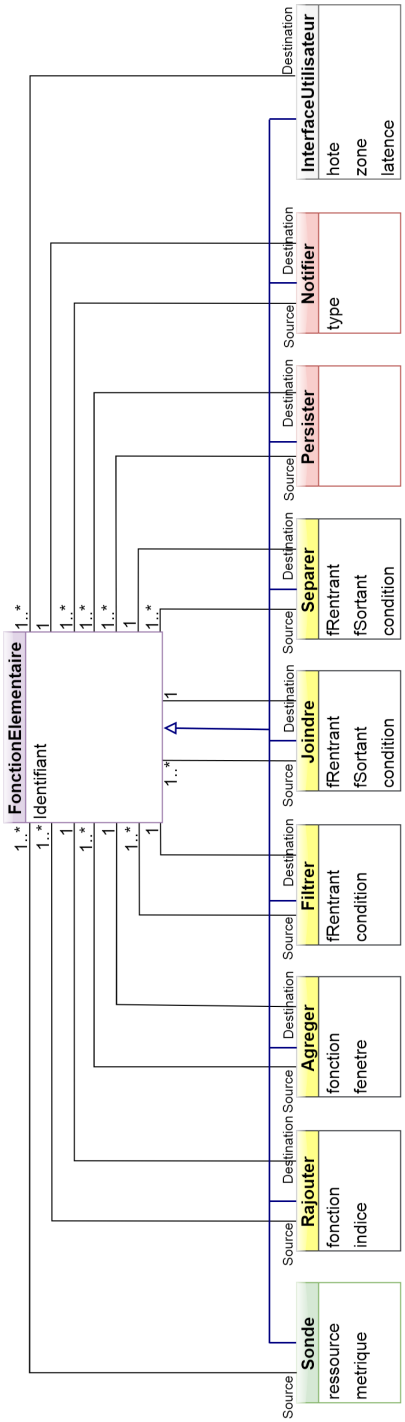

FIGURE 3.2 – Diagramme de classes du langage de description des besoins des utilisateurs

- **Filtrer :** Cette entité supprime tous les n-uplets du flux entrant qui ne satisfont pas une *condition* qui la caractérise. La condition est exprimée en fonction des éléments constituants de chaque n-uplet du *flux entrant*. Elle peut être utilisée, par exemple, pour détecter des dépassements de seuil.

- **Joindre :** À partir de plusieurs flux de mesures entrants, cette fonction génère un seul flux sortant. Les n-uplets du flux sortant résultent de la fusion de ceux des flux entrants. Elle se caractérise par la *condition* qui doit être satisfaite par les n-uplets des flux entrants qui sont fusionnés, ainsi que, par la description des n-uplets du *flux sortant* en fonction de ceux des *flux entrants*.

- **Séparer :** Cette entité applique le traitement réciproque de **joindre**. En effet, à partir d'un seul flux de mesures entrant, elle produit plusieurs flux de mesures sortants. Elle se caractérise par la description des n-uplets des *flux sortants* en fonction de ceux du *flux entrant*.

Nous définissons deux autres entités qui constituent des fonctions d'exposition élémentaires :

- **Persister :** Cette entité enregistre le flux de mesures qu'elle reçoit en entrée. Elle permet de garder une trace sur les états des ressources supervisées.

- **Notifier :** Pour chaque n-uplet du flux qu'elle reçoit, cette entité déclenche un événement (par exemple, envoi d'un courriel ou un SMS à un utilisateur, appel d'une API d'une application) dont le *type* est donné en paramètre.

La dernière entité que nous définissons est l'**interface utilisateur**. Elle représente une destination (par exemple, tableau de bord de supervision) à laquelle il faut exposer les flux de mesures traitées. Elle se caractérise par sa ressource *hôte*, la *latence* maximale tolérée entre sa ressource hôte et celles des **sondes** à l'origine des flux reçus et par la *zone de confiance* qui peut héberger le traitement de l'utilisateur.

3.2.2 Grammaire

Pour exprimer leurs besoins de supervision, les utilisateurs doivent mentionner les entités dont ils ont besoin. Pour chaque entité requise, ils doivent spécifier : son identifiant unique, sa nature et ses caractéristiques. En plus, ils doivent mentionner les flux échangés entre les entités pour réaliser le traitement requis. Pour cela, ils doivent spécifier pour chaque entité les sources et destinations de ses flux de telle façon que le nombre de sources et destinations de chaque entité est respecté. Ce nombre dépend de la nature de l'entité comme l'illustre le diagramme de classe de la figure 3.2.

3.3 Langage de description de l'infrastructure

Une variété de langages a été proposée pour décrire les infrastructures IT et réseau. Cependant, aucun de ces langages n'est destiné pour l'infrastructure Edge. En plus, ces

langages ont été conçus pour répondre aux besoins de différents services qui peuvent être hébergés sur l'infrastructure. Pour satisfaire toutes leurs exigences, ils nécessitent la description de plusieurs aspects dont certains ne font pas partie du cadre de notre intérêt. Pour ces raisons, nous optons pour l'utilisation d'un langage destiné à notre besoin. Nous le positionnons par rapport aux langages existants dans la section 3.3.3.

Le langage que nous utilisions est illustré par un diagramme de classes dans la figure 3.3. Chaque classe représente une entité du lexique. Les attributs de la classe sont des caractéristiques à spécifier pour chaque entité. Les associations entre les classes sont définies selon les règles du langage. Un exemple d'utilisation de ce langage est illustré dans la section 3.5.

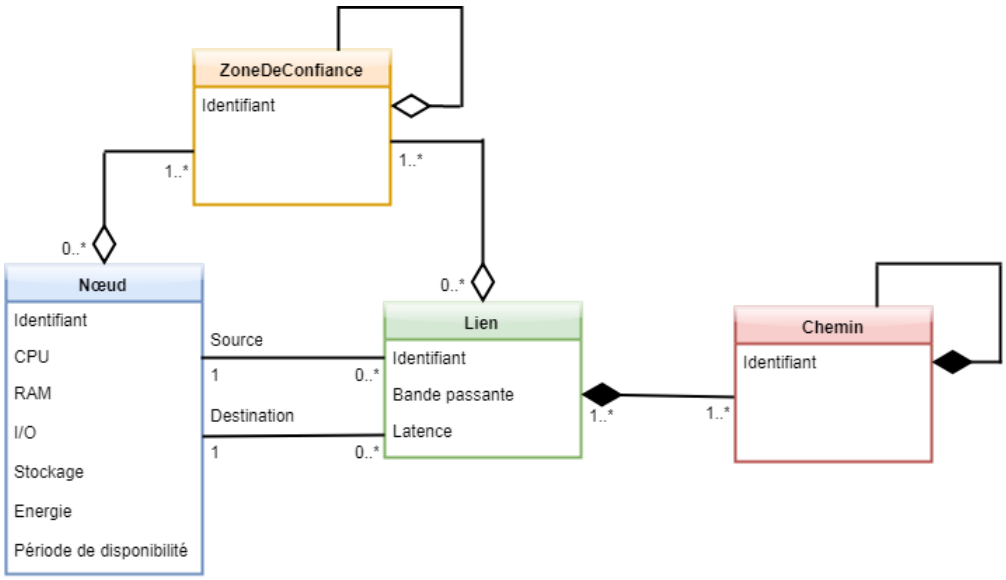

FIGURE 3.3 – Diagramme de classes du langage de description de l'infrastructure

3.3.1 Lexique

Le lexique du langage que nous proposons se compose de quatre entités.

- **Nœud :** Cette entité représente un nœud de calcul (par exemple, machine virtuelle, ordinateur portable, téléphone portable) ou un nœud réseau (par exemple, routeur, commutateur, passerelle domestique). Nous représentons les deux types par la même entité car dans l'infrastructure Edge, la même ressource peut jouer les deux rôles en même temps. À titre d'exemple, un serveur peut jouer le rôle d'un commutateur ou d'un routeur lorsqu'il héberge les fonctions de réseau virtuel vSwitch ou vRouter. Inversement, un commutateur ou un routeur peut jouer le rôle d'un serveur s'il supporte le paradigme du Edge Computing (*c.-à-d.*, il peut héberger du calcul).

  Pour placer les fonctions de supervision en satisfaisant les contraintes de QoS, il faut considérer la capacité d'hébergement du nœud. Nous la caractérisons par

quatre propriétés : le nombre de cœurs du processeur, la RAM, les opérations I/O par seconde et le stockage. En plus, étant donné que les ressources à la périphérie du réseau peuvent avoir une énergie limitée, nous considérons l'énergie disponible au niveau de chacune. Enfin, les ressources de l'infrastructure peuvent ne pas être dédiées à l'opérateur et offertes pour une durée limitée par les utilisateurs. Pour cela, nous considérons la période durant laquelle elles sont disponibles.

- **Lien :** Le **lien** est la ressource qui permet la communication entre deux **nœuds** différents. Un **lien** peut être filaire (par exemple, cuivre, fibre) ou sans fil (par exemple, Wi-Fi).

  Nous caractérisons la capacité d'hébergement du **lien** par deux propriétés, à savoir la bande passante dont il dispose et la latence qu'il introduit. Cette dernière propriété est primordiale pour considérer la distribution massive de l'infrastructure Edge dans le placement des services. Nous caractérisons les liens également par leurs directions car ils peuvent être asymétriques. À titre d'exemple, la bande passante de chargement diffère généralement de la bande passante de téléchargement.

- **Chemin :** Le **chemin** est une chaîne de **liens** qui forment un lien logique. Il permet ainsi la communication entre deux **nœuds** qui ne sont pas connectés par un seul **lien**. Il se caractérise par sa direction qui est la résultante des directions des **liens** qui le forment.

- **Zone de confiance :** La **zone de confiance** représente un ensemble de ressources offertes par le même tenant ou par un ensemble de tenants qui ont les mêmes caractéristiques du point de vue sécurité.

3.3.2 Grammaire

Chaque entité de l'infrastructure doit être identifiée en spécifiant sa nature, son identifiant unique et ses propriétés caractéristiques. En plus, chacune doit vérifier des règles spécifiques en fonction de sa nature. Ces règles concernent ses relations avec les autres entités :

- **Règles spécifiques au nœud :** Chaque **nœud** doit appartenir au moins à une **zone de confiance**.

- **Règles spécifiques au lien :** Pour chaque **lien**, il faut spécifier les deux **nœuds** qu'il relie. Comme nous considérons des liens orientés, il faut spécifier le **nœud** qui représente la source et celui qui représente la destination. En plus, comme c'est le cas des **nœuds**, chaque **lien** doit appartenir au moins à une **zone de confiance**.

- **Règles spécifiques au chemin :** La définition des **chemins** doit respecter les règles de routage (par exemple, il ne doit pas contenir de cycle). Chaque **chemin** peut être construit de deux manières. Premièrement, il peut être constitué d'un seul lien. Deuxièmement, il peut être constitué d'un chemin et d'un lien qui le prolonge (*c.-à-d.*, le nœud destination du chemin est le nœud source du lien). Cette

règle récursive permet de vérifier que les liens qui composent les chemins forment une chaîne. En plus, elle unifie la manière avec laquelle les chemins sont définis et permet ainsi d'éviter la définition de chemins redondants. Cela est nécessaire pour ne pas augmenter significativement le temps de calcul des mutualisations étant donné que ce problème est NP-complet comme nous le détaillons dans la Section 4.1.2.

- **Règles spécifiques à la zone de confiance :** Les entités de type **zone de confiance** peuvent faire partie les unes des autres. Il n'y a pas de contraintes particulières sur les relations entre ces entités. En effet, une même zone peut avoir plusieurs sous-zones (par exemple, une zone qui représente un pays peut avoir plusieurs sous-zones qui représentent des infrastructures d'opérateurs différents). Réciproquement, une zone peut être elle-même une sous-zone de plusieurs zones simultanément (par exemple, un service d'hébergement PaaS qui s'appuie sur l'infrastructure de plusieurs opérateurs peut représenter une sous-zone des zones qui représentent les infrastructures de ces opérateurs).

3.3.3 Comparaison avec des langages existants

Le "Network Markup Language" (NML) [Ham et al. 2013] et le "Infrastructure and Network Description Language" (INDL) [Ghijsen et al. 2013] figurent parmi les langages les plus utilisés pour la description des infrastructures IT et réseau. Nous comparons ces deux langages avec celui que nous proposons.

Comparaison avec NML

NML est un langage destiné à la description fonctionnelle des infrastructures réseau qui peuvent se composer de différents niveaux (par exemple, réseau physique, réseau virtuel) et appartenir à plusieurs domaines.

Les entités sur lesquelles s'appuie ce langage ainsi que les relations entre elles sont représentées par un diagramme de classes dans la figure 3.4. Les entités **node** et **link** sont équivalentes aux entités **nœud** et **lien** de notre langage. L'entité **port** permet de prendre en considération le niveau auquel appartient le réseau (par exemple, réseau physique, réseau virtuel). Nous ne considérons pas cette entité dans notre langage car il n'est pas nécessaire pour le placement. L'entité **topology** se compose d'un ensemble chaîné d'éléments **node**, **link** et **port**. Elle est équivalente, dans notre langage, à l'entité **chemin** qui, par contre, ne nécessite pas la spécification d'une entité équivalente à **port** dans les chaînes. L'entité **topology** constitue un assemblage spécifique des entités **node**, **link** et **port**. Tous les assemblages possibles sont représentés par l'entité **group**. L'entité **service** représente les fonctionnalités possibles des ressources. À titre d'exemple, ces fonctionnalités peuvent être le routage de données (**switching service**), l'encodage de données (**adaptation service**) ou le décodage de données (**deadaptation service**). Pour notre besoin, toutes les ressources ont une seule fonctionnalité qui est l'hébergement du service. Pour cette raison, nous ne considérons pas cette entité. L'entité **location**

représente l'emplacement géographique où se trouve la ressource. Dans notre contexte, cette information se retrouve dans la latence des liens et la zone de confiance que nous considérons dans notre langage. L'entité **lifetime** représente la période de disponibilité des ressources. Elle figure, dans notre langage, en tant qu'une propriété des **nœuds**. Les entités **label**, **label group** et **ordered list** permettent de structurer les relations entre les autres entités.

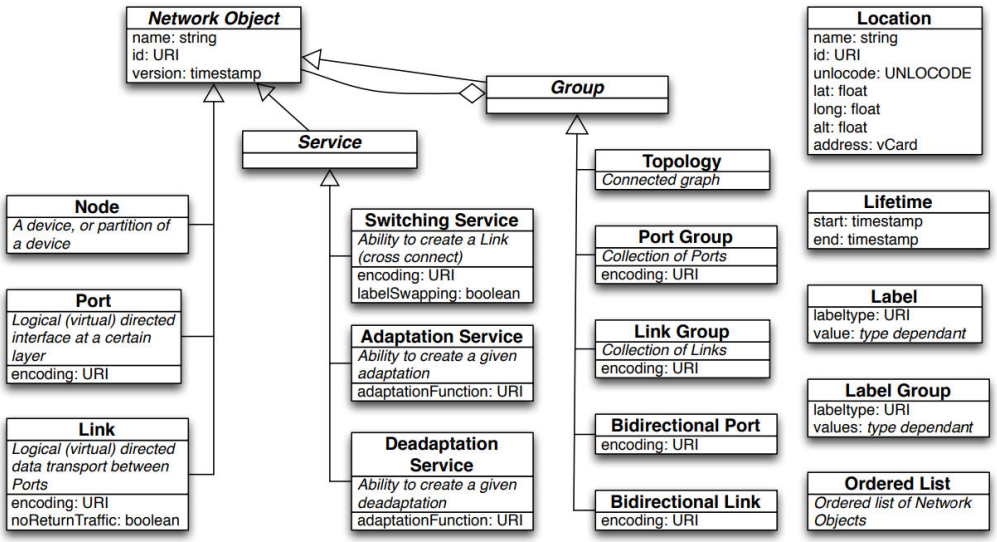

FIGURE 3.4 – Diagramme de classes de NML [HAM et al. 2013]

Comparaison avec INDL

INDL est un langage destiné à la description des ressources de calcul des infrastructures Cloud. Il peut être intégré nativement avec NML. Cela permet de décrire aussi les ressources réseau de l'infrastructure Cloud.

L'entité **node** représente une ressource de calcul. Comme l'illustre la figure 3.5, elle est caractérisée par ses capacités de stockage, son processeur et sa mémoire qui sont regroupés dans l'entité **node component**. L'utilisateur peut définir des entités équivalentes à **node component** pour ajouter d'autres caractéristiques. Dans le langage que nous proposons, nous avons considéré, en plus des caractéristiques proposées par NML, les I/O et l'énergie disponible car elles impactent le placement. INDL s'appuie sur l'entité **virtual node** pour décrire une machine virtuelle. Comme l'illustre la figure 3.6, cette entité hérite les caractéristiques de l'entité **node** (*c.-à-d.*, stockage, processeur et mémoire). Elle peut être implémentée aussi par l'entité **node**. Cela permet d'imbriquer les couches de virtualisation. Dans le langage que nous proposons, nous ne représentons pas la virtualisation car elle n'a pas d'impact sur le placement.

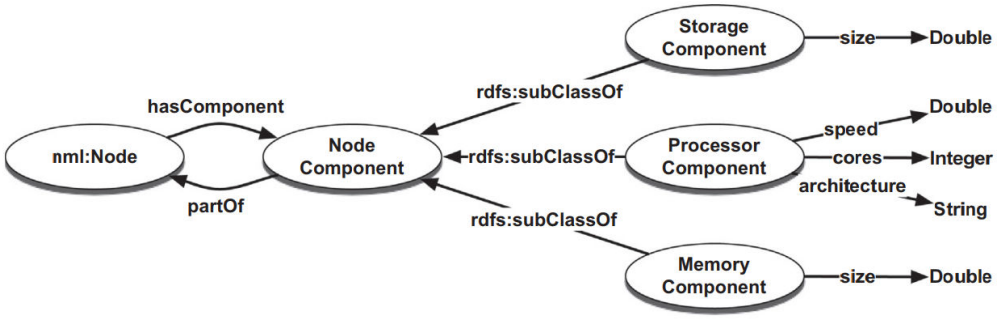

FIGURE 3.5 – Schéma RDF d'un nœud INDL [GHIJSEN et al. 2013]

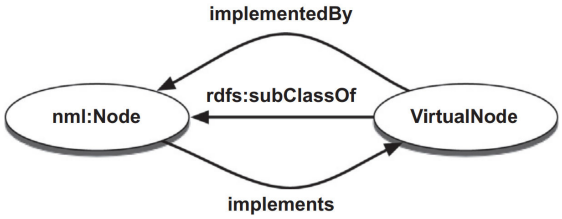

FIGURE 3.6 – Schéma RDF de la virtualisation avec INDL [GHIJSEN et al. 2013]

3.4 Calculateur de placement

Les deux langages présentés dans les sections précédentes permettent de décrire les besoins des utilisateurs et les capacités de l'infrastructure que le **calculateur de placement** reçoit en entrée. Cet élément est chargé de déterminer l'architecture de déploiement des fonctions de supervision. Pour vérifier que cette architecture satisfait les besoins fonctionnels et de qualité de service des utilisateurs, il calcule son placement sur l'infrastructure.

Le placement peut être calculé en vue d'atteindre plusieurs objectifs de performance. Dans cette thèse, nous avons choisi de nous concentrer sur la minimisation de l'empreinte de calcul et réseau introduite par le service de supervision sur l'infrastructure. En effet, l'empreinte d'une supervision holistique de l'infrastructure Edge peut être significative. Par exemple, les sondes utilisées pour la supervision de l'infrastructure de Twitter génèrent 2.8 milliards de mesures par minute [ASTA 2016]. Un seul capteur qui supervise la consommation d'énergie électrique génère 50 milliards de mesures chaque année [ANDERSEN et al. 2016]. Pour ne pas surcharger l'infrastructure, nous prenons en considération l'empreinte de la supervision lors du calcul du placement.

Pour réduire l'empreinte du service de supervision, nous proposons de mutualiser les traitements et les flux identiques qui sont requis par des utilisateurs différents. L'existence de tels traitements et flux est favorisée par l'aspect multi-tenant de l'infrastructure Edge. En effet, les tenants qui partagent les mêmes ressources peuvent avoir les mêmes besoins (en totalité ou partiellement) pour les superviser. La section suivante montre la pertinence de notre approche dans un cas d'utilisation.

3.5 Cas d'utilisation

Le cas d'utilisation que nous considérons pour illustrer les langages de description ainsi que notre approche de placement est un réseau de diffusion de contenu [Pathan et al. 2007]. C'est un service d'hébergement distribué qu'offrent les opérateurs d'infrastructures. Les utilisateurs typiques de ce service sont les fournisseurs de vidéos à la demande (VoD). Il leur permet d'héberger leurs contenus multimédia (par exemple, vidéo, images) à proximité de leurs clients finaux et d'améliorer ainsi l'expérience de l'utilisateur. L'opérateur de l'infrastructure Edge, peut s'appuyer sur les ressources à la périphérie du réseau dans l'offre de ce service. Cela permet de réduire la latence, l'utilisation de la bande passante et le coût énergétique [Wang et al. 2017].

Pour offrir ce service à la périphérie du réseau, trois types de ressources peuvent être utilisés : la ressource hôte à l'Edge (par exemple, passerelle résidentielle), le contenu multimédia et l'environnement d'hébergement des contenus multimédia. Cet environnement d'hébergement (par exemple, Docker [2], LXC [3]) est une couche qui assure l'isolation des contenus multimédia du reste des processus de la ressource hôte. Il est administré par l'opérateur de l'infrastructure pour gérer le placement des contenus.

3.5.1 Description des besoins de supervision de l'opérateur

Afin d'illustrer l'utilisation du **langage de description des besoins des utilisateurs**, nous décrivons les besoins de supervision de l'opérateur de l'infrastructure dans une configuration où deux ressources hôtes à la périphérie du réseau sont mobilisées comme l'illustre la figure 3.7.

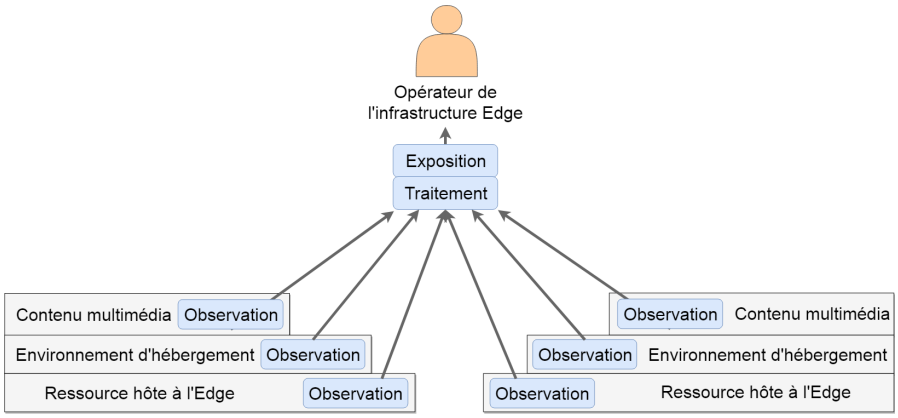

Figure 3.7 – Cas d'utilisation

Nous supposons que l'opérateur a besoin de recevoir une notification toutes les 10 s (avec un retard maximal de 18 ms) si l'environnement d'exécution consomme plus de 90% de la RAM offerte par la passerelle hôte. En plus, il requiert de garder une trace sur

[2] https ://www.docker.com/ (visité le 01/10/2018)
[3] https ://www.docker.com/ (visité le 01/10/2018)

ces notifications. Il a besoin de garder une trace également sur l'utilisation des contenus multimédia en RAM pour identifier ceux qui consomment le plus de RAM et qui doivent être migrés en priorité en cas de besoin. Enfin, nous supposons que les sondes disponibles au niveau de toutes les ressources remontent une mesure par seconde.

Nous représentons sur la figure 3.8, le diagramme d'objets qui décrit ce besoin de supervision. Les objets *s1*, *s2*, *s3*, *s4*, *s5* et *s6* de type **Sonde** permettent d'exprimer le besoin d'observer les différentes mesures de RAM. Les remontées étant réalisées toutes les secondes, les objets *a1*, *a2*, *a3*, *a4*, *a5* et *a6* de type **Agréger** calculent les moyennes des mesures sur une fenêtre de 10 s. Les objets *j1* et *j2* alignent sur le même intervalle temporel les mesures de la RAM utilisée par les environnements d'hébergement et les mesures de la RAM disponible au niveau des passerelles résidentielles. Les objets *f1* et *f2* de type **Filtrer** détectent si l'environnement d'exécution consomme plus de 90% de la RAM disponible au niveau de sa passerelle hôte. Le cas échéant, les objets *n1* et *n2* de type **Notifier** font appel à l'entité *iu1* de type **InterfaceUtilisateur**. Les notifications sont tracées par les objets *p1* et *p2* de type **Persister**. Quant à l'utilisation de la RAM par les contenus multimédia, elle est tracée par les objets *p3* et *p4* de type **Persister** également.

3.5.2 Description de l'infrastructure

Pour illustrer l'utilisation du **langage de description de l'infrastructure**, nous décrivons un exemple d'une infrastructure qui peut héberger un réseau de diffusion de contenu à la périphérie du réseau. Nous supposons que cette infrastructure est constituée de deux PoPs et de deux passerelles résidentielles à la périphérie du réseau telles que chaque passerelle est connectée aux deux PoPs.

Nous représentons dans la figure 3.9, le diagramme d'objets qui décrit cette infrastructure. Afin de le simplifier, nous ne représentons pas les objets des classes **Chemin** et **ZoneDeConfiance**. Les objets *s1* et *s2* représentent les deux PoPs. Nous représentons chaque PoP par un seul **Nœud** car la latence entre les serveurs au sein des PoPs modernes est négligeable (inférieure à 1 ms [Rumble et al. 2011]). Les objets *pass1* et *pass2* représentent les passerelles résidentielles. Les objets *i1*, *i2* et *i3* représentent les serveurs qui hébergent les objets de type **InterfaceUtilisateur**. Ils ne peuvent pas être exploités pour héberger les fonctions de supervision. C'est pour cette raison, que leurs capacités d'hébergement sont nulles. Les capacités d'hébergement du reste des objets de types **Nœud** et **Lien** ont été choisies à titre d'illustration et en prenant en considération les différences de performances relatives entre les différentes ressources de l'infrastructure comme détaillé dans la section 1.3.2.

3.5.3 Placement mutualisé des fonctions de supervision

Dans ce cas d'utilisation, il est possible de réduire l'empreinte de la supervision considérablement en mutualisant le traitement et les flux entre les différents tenants. En effet, les trois types de ressources utilisées (*c.-à-d.*, la ressource hôte à l'Edge, le contenu multimédia et l'environnement d'hébergement des contenus multimédia) ont des tenants

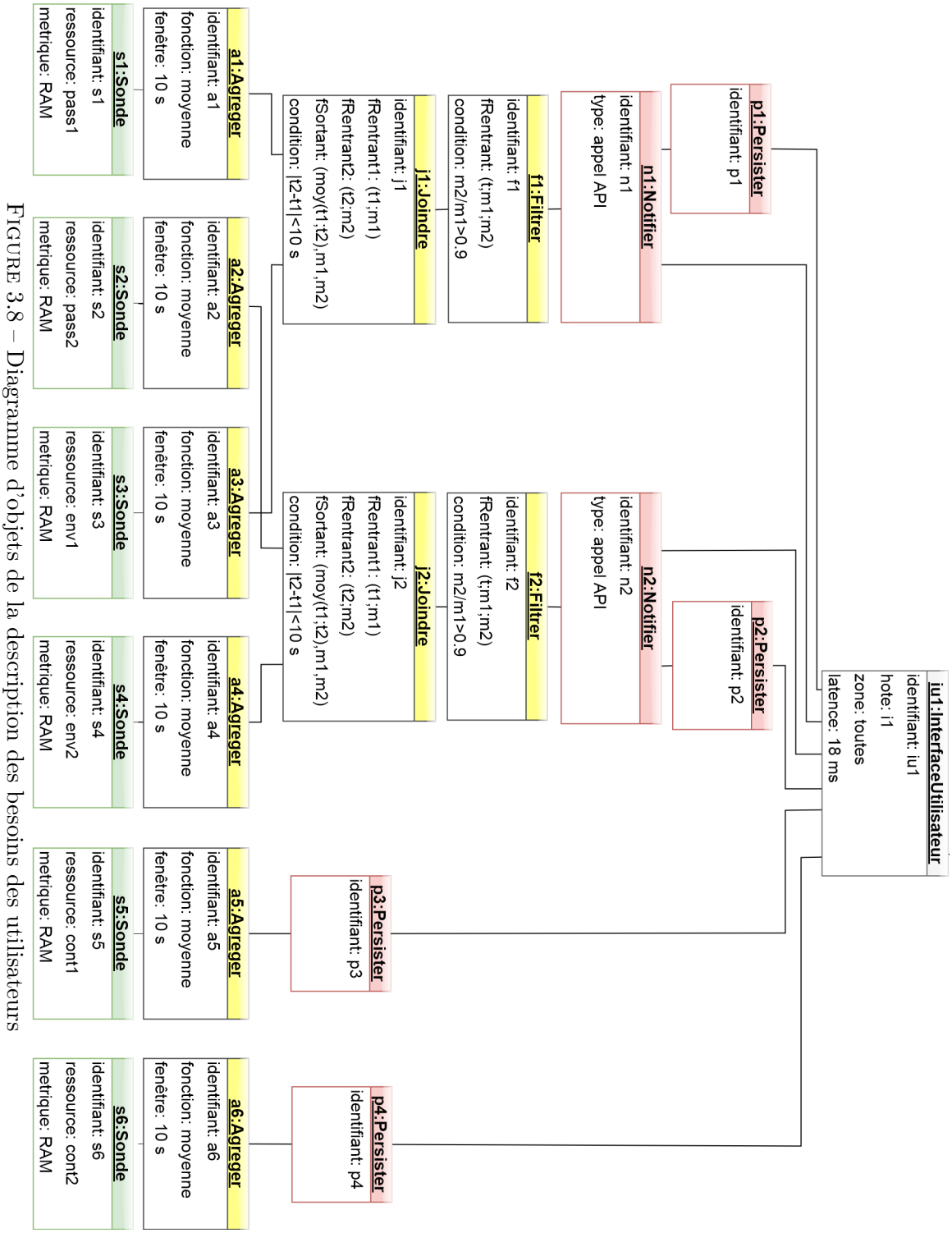

FIGURE 3.8 – Diagramme d'objets de la description des besoins des utilisateurs

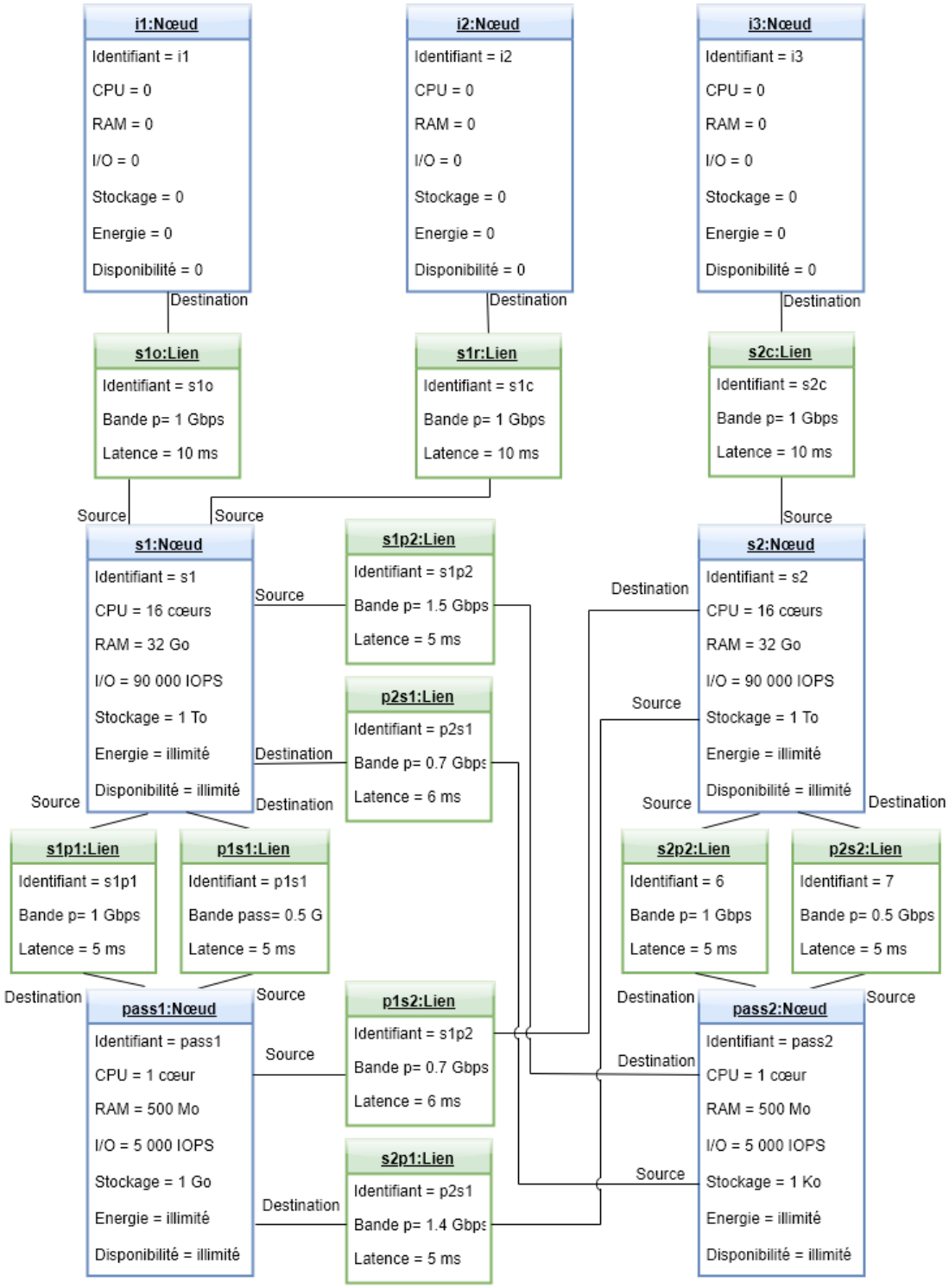

FIGURE 3.9 – Diagramme d'objets de description des besoins des utilisateurs

différents qui peuvent avoir des besoins communs de supervision. Un exemple est illustré par la table 3.1. Le fournisseur de contenus requiert de superviser l'utilisation de ses contenus multimédia pour facturer ses clients. Le fournisseur de la ressource hôte à l'Edge requiert de superviser les capacités disponibles de ses ressources et celles utilisées par l'environnement d'hébergement. L'opérateur de l'infrastructure, quant à lui, requiert de superviser l'ensemble des trois métriques (*c.-à-d.*, les capacités disponibles au niveau des ressources hôtes, les capacités utilisées par l'environnement d'hébergement et les capacités utilisées par le contenu) comme détaillé dans la section 3.5.1.

		Tenants		
		Fournisseur de contenus	Fournisseur de ressources à l'Edge	Opérateur de l'infrastructure Edge
Métriques	Capacités utilisées par le contenu multimédia	✓		✓
	Capacités utilisées par l'environnement d'hébergement		✓	✓
	Capacités disponibles de la ressource hôte à l'Edge		✓	✓

TABLE 3.1 – Les métriques qui intéressent chaque tenant

Nous supposons que le traitement de supervision requis par le fournisseur de ressources à l'Edge et le fournisseur de contenus consistent à agréger les mesures remontées par les sondes toutes les 10 secondes. La figure 3.10 (a) illustre la configuration native où chaque tenant est servi par des fonctions de traitement qui lui sont dédiées (elles sont représentées par la même couleur que leur tenant). La figure 3.10 (b) illustre la configuration que nous proposons et qui consiste à mutualiser l'agrégation des mesures entre les différents tenants. Les différences entre les deux configurations (a) et (b) de la figure 3.10 montrent l'importance de notre approche dans la réduction de l'empreinte de la supervision. D'une part, elle réduit l'empreinte réseau car les flux de mesures ne sont envoyés qu'une seule fois pour les besoins de tous les tenants. D'autre part, elle réduit l'empreinte de calcul puisqu'une partie du traitement est mutualisée.

La pertinence de la mutualisation ne se limite pas au cas d'utilisation précédent. Elle peut être illustrée par les différents services de l'infrastructure Edge pour lesquels différents tenants peuvent avoir des intérêts communs de supervision. Par exemple, dans l'offre Network-as-a-Service [PRIES et al. 2016], l'opérateur partage ses ressources avec les utilisateurs qui deviennent à leur tour des tenants de ces ressources. Par conséquent, la supervision des ressources partagées est requise par les utilisateurs ainsi que l'opérateur. Les services qui s'appuient sur l'IdO peuvent également illustrer la pertinence de notre approche. Par exemple, les réseaux électriques intelligents [FANG et al. 2012] est un service dont l'objectif est d'améliorer la distribution de l'électricité. Pour cela, il s'appuie sur des capteurs connectés qui mesurent la consommation et la production de cette énergie. Les mesures remontées par ces capteurs intéressent plusieurs intervenants comme les services de production, de transmission, d'exploitation et de maintenance ainsi que les consommateurs finaux (c.f. annexe B de [GREER et al. 2014]).

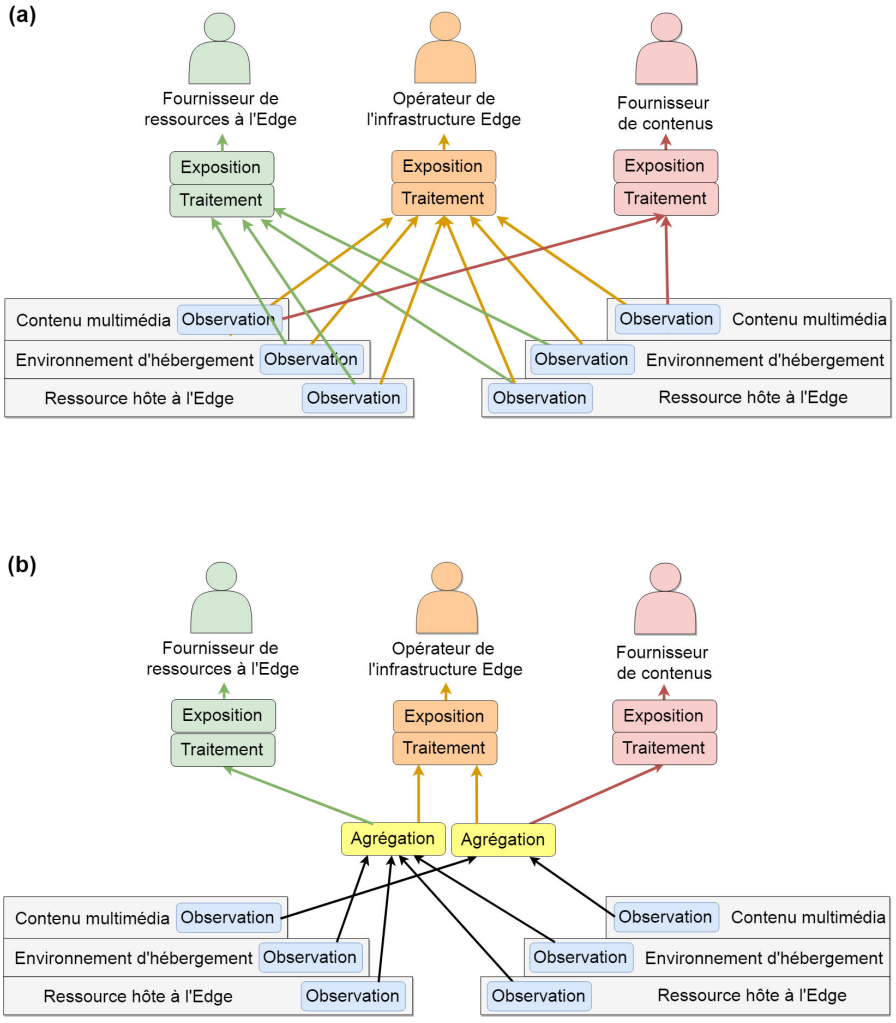

FIGURE 3.10 – Mutualisation des fonctions de supervision

Cependant, il faut faire face à deux défis qui rendent l'opération de mutualisation complexe :

Identification du traitement et des flux identiques

L'identification du traitement et des flux identiques qui peuvent être mutualisés à partir de la description des besoins de supervision n'est pas évidente. Cela est dû essentiellement à deux raisons. D'une part, le traitement commun peut être un traitement élémentaire d'une fonction requise par un utilisateur. Par exemple, le fournisseur de contenus multimédia peut avoir besoin d'agréger les mesures du taux d'utilisation des contenus sur une fenêtre de 15 secondes tandis que l'opérateur a besoin d'agréger ces mesures sur une fenêtre de 10 secondes. Dans ce cas, le traitement commun qui consiste à agréger les mesures toutes les 5 secondes représente un traitement élémentaire des deux fonctions d'agrégation requises par ces utilisateurs. D'autre part, le traitement commun peut être difficile à identifier lorsqu'il est exprimé sur des flux différents. Par exemple, dans l'expression des besoins de supervision de l'opérateur représentée sur la figure 3.8, l'identification du traitement d'agrégation commun avec le fournisseur de contenus n'aura pas été facile si l'agrégation était réalisée sur le flux sortant de la fonction jointure et non pas sur son flux entrant.

Pour faire face à ce défi, il faut analyser les besoins de supervision des utilisateurs et les optimiser avec deux opérations. D'abord, il faut effectuer l'opération « séparation » [Hirzel et al. 2014] pour exprimer le traitement nécessaire dans la granularité la plus fine. Par exemple, une fonction qui agrège les mesures toutes les 10 secondes peut être subdivisée en deux fonctions successives telles que l'une agrège les mesures toutes les 5 secondes et l'autre agrège les mesures résultantes toutes les 10 secondes. Ensuite, il faut effectuer l'opération « réorganisation » [Hirzel et al. 2014] pour prioriser l'exécution du traitement qui est requis par la plupart des utilisateurs. Des implémentions de ces optimisations existent comme [Yu et al. 2009] pour l'opération « séparation » et [Arasu et al. 2006] pour l'opération « réorganisation ».

Distribution de l'infrastructure

Étant donné la distribution massive de l'infrastructure Edge, il est possible que les tenants qui requièrent le même traitement soient distants les uns des autres. Par conséquent, la mutualisation globale (en un seul endroit) de leur traitement peut empêcher la satisfaction de leurs exigences en latence. Par exemple, supposons que la fonction **InterfaceUtilisateur** du fournisseur de contenu doit être placée sur le nœud *i3* du diagramme objet de la figure 3.9, que le fournisseur de contenu n'autorise pas une latence supérieure à 18 ms et que sa fonction d'agrégation de l'utilisation des contenus au niveau de la passerelle *pass2* nécessite des performances de stockage supérieures à celles disponibles au niveau de cette passerelle. Dans ce cas, la mutualisation de cette fonction d'agrégation entre le fournisseur de contenu et l'opérateur n'est pas possible. En plus, dans certains cas, la mutualisation peut augmenter l'empreinte sur le réseau. C'est le cas lorsque le traitement mutualisé doit être placé plus loin de certains utilisateurs pour satisfaire les exigences de latence d'autres utilisateurs. Par exemple, nous pouvons considérer le cas précédent

en supposant que la latence autorisée par le fournisseur de contenu est 28 ms. Dans ce cas la mutualisation de sa fonction d'agrégation avec celle de l'opérateur est possible sur le nœud *s1*. Cependant, cela augmente l'empreinte de la supervision sur le réseau car il requiert d'utiliser les liens *s1p2* et *p2s1* ce qui n'est le cas si le traitement n'est pas mutualisé.

Pour faire face à ce défi, il faut considérer la distribution de l'infrastructure lors du calcul des mutualisations. Pour cela, nous formalisons, dans le chapitre 4 le calcul du placement mutualisé en tant qu'un problème de satisfaction de contraintes qui prend en considération les besoins fonctionnels et de qualité de service des utilisateurs ainsi que les capacités et la topologie de l'infrastructure.

3.6 Conclusion

Dans ce chapitre, nous avons décrit un canevas logiciel pour la mise en place d'un service de supervision qui est géré par une boucle de contrôle autonomique MAPE. Cette boucle s'appuie sur une fonction de gestion qui se compose d'un **ordonnanceur de placement** et d'un **calculateur de placement**. La conception de ces deux modules représente deux défis bien distincts ; nous nous sommes concentrés sur le **calculateur de placement**. Le rôle de ce dernier consiste à déterminer l'architecture de déploiement qui satisfait les besoins de supervision des utilisateurs. Pour faire face à la diversité de ces besoins, nous avons défini un **langage** qui unifie leur description. Afin de vérifier que le service de supervision satisfait les besoins de qualité de service des utilisateurs, le **calculateur de placement** nécessite une description des capacités et de la topologie de l'infrastructure. Pour adresser l'hétérogénéité de l'infrastructure Edge, nous avons défini un **langage** qui unifie sa description. Le placement qui satisfait les besoins des utilisateurs peut être calculé en vue d'atteindre plusieurs objectifs de performance. Étant donnée l'importante empreinte de calcul et réseau que risque d'introduire une supervision holistique de l'infrastructure Edge, nous avons choisi de nous concentrer sur la minimisation de cette empreinte. Pour cela, nous avons proposé de mutualiser les flux et les fonctions identiques entre des tenants différents. Nous avons illustré l'utilisation des langages que nous avons décrits et l'approche que nous avons proposée pour réduire l'empreinte du service de supervision dans un cas d'utilisation. Cela nous a permis de souligner que la distribution de l'infrastructure Edge est un aspect important qui doit être pris en considération lors du calcul du placement mutualisé.

Dans le chapitre suivant, nous formalisons le calcul du placement mutualisé en tant qu'un problème de satisfaction de contraintes qui prend en considération la distribution de l'infrastructure Edge.

4

Formalisation du problème de placement mutualisé

Sommaire

Le calcul des mutualisations des fonctions de supervision est une étape primordiale dans notre proposition. Dans ce chapitre, nous le formalisons. Nous commençons par présenter le problème à résoudre en établissant le lien avec les travaux existants et en introduisant notre approche de modélisation. Ensuite, nous exposons notre modèle. Enfin, nous évaluons la capacité du modèle à réduire l'empreinte de supervision en considérant le coût qu'il introduit en termes de temps de calcul.

4.1 Présentation du problème

À notre connaissance, nous sommes les premiers à formaliser la mutualisation des traitements et des flux identiques entre des tenants différents afin de réduire l'empreinte de supervision de l'infrastructure Edge. Nous modélisons notre approche par un problème de satisfaction de contraintes.

4.1.1 Travaux existants

Le partage des sondes entre différents tenants a été identifié dans la littérature de l'IdO comme une opportunité pour augmenter la valeur du métier [Jernigan et al. 2016]. Afin de promouvoir ce partage, différentes études ont été menées. Par exemple, une architecture à trois couches a été proposée pour faire face à l'hétérogénéité des objets connectés [Benazzouz et al. 2014], un modèle de méta-données a été proposé pour unifier la description des observations remontées par les objets connectés [Xu et al. 2014a] et des méthodes formelles ont été proposées pour identifier la similarité entre les objets connectés [Antunes et al. 2018]. Ces études s'intéressent au partage des mesures brutes mais n'examinent pas le partage du traitement réalisé sur ces mesures.

Le traitement des flux de données tout en tenant compte des capacités de l' infrastructure Edge a été récemment étudié dans la littérature dans le contexte de l'exploration de données. Dans [Assunção et al. 2018], les auteurs étudient la capacité de l' infrastructure Edge à réaliser le traitement des flux de données à proximité des utilisateurs. Dans [Cardellini et al. 2017], les auteurs formalisent le problème du placement de processeurs de flux dans une infrastructure Edge afin de répondre aux besoins des utilisateurs en considérant les capacités de l'infrastructure. Cependant, ces travaux ne considèrent pas la possibilité de mutualiser le traitement entre différents tenants.

La mutualisation est présente partiellement dans le problème de chaînage de fonctions réseau [Halpern et al. 2015]. Ce problème consiste à router efficacement les flux réseau des utilisateurs à travers une chaîne de fonctions réseau (par exemple, pare-feu, systèmes de détection d'intrusion, système d'équilibrage de charge) pour réduire l'empreinte globale du service réseau. Dans ce contexte, des études comme [Luizelli et al. 2015] et [Tastevin et al. 2017] proposent des modèles où la même fonction réseau est dédiée au traitement de différents flux d'utilisateurs. Ainsi, les ressources hôtes sont exploitées davantage et leur temps d'inactivité est réduit. Cependant, ces travaux ne considèrent pas spécifiquement le cas des utilisateurs ayant des flux identiques. Par conséquent, les flux sont traités d'une manière redondante.

4.1.2 Modélisation par un problème de satisfaction de contraintes

Pour réussir la mutualisation, il faut minimiser l'usage des ressources de l'infrastructure tout en satisfaisant les exigences fonctionnelles et de QoS de chaque utilisateur.

Les exigences fonctionnelles des utilisateurs sont les chaînes de fonctions exprimées à l'aide du langage de description des besoins présenté au chapitre 3. Nous les modélisons par un graphe orienté dont les nœuds représentent les fonctions élémentaires et dont les

arcs représentent les flux entre ces fonctions. Nous modélisons la mutualisation par une contraction d'arêtes [Asano et al. 1982] de ce graphe. Cela correspond à fusionner les nœuds d'un graphe selon des contraintes spécifiques. Il a été démontré qu'il s'agit d'un problème NP-complet [Abdulrahim 1998].

Afin de satisfaire les exigences de QoS (par exemple, la latence), il faut placer les fonctions de supervision en prenant en considération les capacités des ressources de l'infrastructure. Nous modélisons l'infrastructure par un graphe dont les nœuds sont les ressources de calcul et les arcs sont les liens qui les relient. Nous modélisons le placement par un couplage de graphes inexacts [Bengoetxea 2002] entre le graphe qui modélise les besoins fonctionnels des utilisateurs et celui qui modélise l'infrastructure. Comme c'est le cas de la contraction des arêtes, le couplage des graphes inexacts est NP-complet [Asano et al. 1982].

L'identification du placement mutualisé est donc un problème NP-complet orienté par les contraintes. Nous optons pour le formaliser en tant qu'un problème de satisfaction de contraintes [Kumar 1992]. Un tel problème peut être exprimé par un triplet (X, D, C) tel que :

- X est l'ensemble des variables. Une variable est une inconnue à déterminer.
- D est l'ensemble des domaines. Chaque domaine est associé à une variable. Il représente l'ensemble de ses valeurs possibles.
- C est l'ensemble de contraintes que chaque élément de X doit satisfaire.

4.2 Modèle du problème de placement mutualisé

Notre modèle est résumé dans la table 4.1 et il sera détaillé dans la suite.

4.2.1 Notions et notation

Afin de modéliser le problème de placement mutualisé, nous définissons quatre notions :

- **Fonction** : Une fonction peut avoir l'un des types présentés dans le chapitre 3 : sonde, traitement et exposition (*c.-à-d.*, rajouter, agréger, filtrer, joindre, séparer, persister, notifier) ou interface utilisateur (par exemple, tableau de bord, base de données).
- **Flux** : Un flux est l'ensemble des données envoyées d'une fonction à une autre (le long d'un chemin de liens qui les relie).
- **Serveur** : Un serveur est une ressource de calcul de l'infrastructure Edge. Il peut héberger des fonctions qui requièrent moins de capacités cumulatives que celles qu'il offre.
- **Lien** : Un lien est une ressource réseau qui connecte les serveurs de l'infrastructure Edge. Il peut héberger des flux qui, d'un côté, requièrent moins de capacités

Entrées	
P	L'ensemble des fonctions de type sonde
M	L'ensemble des fonctions de type traitement ou exposition
U	L'ensemble des fonctions de type interface utilisateur
F	L'ensemble des fonctions de tous les types : $F = P \cup M \cup U$
R	L'ensemble des flux échangés entre les fonctions
l_{ff}	Pour une paire de $F \times F$, cette fonction retourne vrai si les deux éléments du paire sont identiques (*c.-à-d.*, les deux ont le même type et les mêmes paramètres). Sinon, elle retourne faux.
l_{fC}	Pour un élément de F, cette fonction retourne la capacité requise du serveur hôte de cet élément.
l_{fS}	Pour un élément de F, cette fonction retourne le serveur de S qui doit l'héberger ou *null* s'il n'y a pas de contraintes sur le serveur hôte.
l_{fL}	Pour un élément de F, cette fonction retourne la latence maximale tolérée entre cet élément et la sonde.
l_{oC}	Pour un flux de $F \times F$, cette fonction retourne la capacité requise du lien hôte de ce flux.
G_U	Le graphe qui modélise les besoins des utilisateurs. $G_U = (F, R, l_{fC}, l_{ff}, l_{fS}, l_{fL}, l_{oC})$
S	L'ensemble des serveurs
L	L'ensemble des liens réseau
l_{sC}	Pour un serveur de S, cette fonction retourne la capacité de ce serveur.
l_{lC}	Pour un lien de $S \times S$, cette fonction retourne la capacité de ce lien.
l_{lL}	Pour un lien de $S \times S$, cette fonction retourne sa latence.
G_I	Le graphe qui modélise l'infrastructure : $G_I = (S, L, l_{sC}, l_{lC}, l_{lL})$.
Variables et leurs Domaines	
V_M	L'ensemble des fonctions du service de supervision. Son domaine $D_{V_M} \subseteq F$. Il représente les nœuds du graphe G_M qui modélise le service de supervision
A_M	L'ensemble des flux du service de supervision. Son domaine $D_{A_M} \subseteq F \times F$. Il représente les arcs du graphe G_M qui modélise le service de supervision
x_{sf_i}	Le serveur qui héberge $f_i \in F$. L'ensemble composé par ces variables est $X_{SF} = \{x_{sf_i} \mid i \in [\![1; \lvert F \rvert]\!]\}$. Leur domaine est $D_{X_{SF}} = S$.
x_{lo_i}	L'ensemble des liens qui hébergent $o_i \in F \times F$. L'ensemble qui est composé par ces variables est $X_{LO} = \{x_{lo_i} \mid i \in [\![1; \lvert F \times F \rvert]\!]\}$. Leur domaine est $D_{X_{LO}} = chemins(G_I)$.
x_{or_i}	Le flux du service de supervision qui est équivalent au flux $r_i \in R$. L'ensemble formé par ces éléments est $X_{OR} = \{x_{or_i} \mid i \in [\![1; \lvert R \rvert]\!]\}$. Leur domaine est $D_{X_{OR}} \subseteq F \times F$.
Contraintes	
Fonc.	Equations : 4.1, 4.2 et 4.3.
QoS	Equations : 4.4, 4.5, 4.6, 4.7, 4.8 et 4.9.
Chaînage	Equations : 4.10 et 4.11.
Fonction objectif	
X_{fp}	L'empreinte globale du service de supervision (Equation 4.12)

TABLE 4.1 – Résumé du modèle

cumulatives que celles qu'il offre et qui, d'un autre côté, tolèrent plus de temps de latence que celui qu'il introduit.

Nous représentons un graphe orienté et étiqueté G comme un tuple $G = (V, A, l_1, l_2, ..., l_n)$ tel que V est l'ensemble de ses nœuds, A est l'ensemble de ses arcs ($A \subseteq V \times V$) et l_1, l_2, ..., l_n sont ses fonctions d'étiquetage. Une fonction d'étiquetage l_i peut être une fonction d'étiquetage de nœuds $l_i : V \rightarrow L_i$ ou une fonction d'étiquetage d'arcs $l_i : A \rightarrow L_i$ tel que L_i est l'ensemble des étiquettes des nœuds ou arcs, respectivement.

Enfin, nous définissons les opérateurs suivants :

- $chemins(graphe)$: Pour un graphe donné, cet opérateur retourne tous les chemins formés par ses arcs.
- $source(arcs)$ et $dest(arcs)$: Pour un ensemble donné d'arcs qui forment un chemin, ces opérateurs retournent respectivement le nœud source et le nœud destination du chemin.
- $tête(arc)$ et $queue(arc)$: Pour un arc donné, ces opérateurs retournent respectivement le nœud tête et le nœud queue.

4.2.2 Entrées du problème

Les entrées du problème sont les besoins de supervision des utilisateurs et la description de l'infrastructure. Nous modélisons chacun d'eux par un graphe orienté.

Modélisation des besoins de supervision des utilisateurs

Nous modélisons les besoins des utilisateurs par le graphe orienté $G_U = (F, R, l_{fC}, l_{fS}, l_{fL}, l_{oC}, l_{ff})$. La figure 4.1 illustre un exemple de ce graphe. L'ensemble des nœuds F de G_U représente les éléments du langage de description des besoins des utilisateurs : sondes, fonctions de traitement ou d'exposition et interfaces utilisateur. Soient P, M et U les ensembles qui représentent ces trois catégories d'éléments, respectivement. Ainsi, $F = P \cup M \cup U$. L'ensemble d'arcs R de ce graphe représente les flux requis entre les différentes fonctions. Nous associons quatre fonctions d'étiquetage de nœuds à ce graphe : l_{fC} retourne la capacité d'hébergement requise par une fonction de supervision donnée en paramètre. l_{fS} retourne le serveur sur lequel doit être hébergée une fonction de supervision donnée en paramètre et retourne *null* si la fonction n'a pas de serveur hôte prédéfini. Les utilisateurs et les sondes sont les fonctions qui peuvent avoir des serveurs hôtes prédéfinis, car dans certains cas, ils doivent être hébergés sur le serveur hôte de l'utilisateur ou sur la ressource observée, respectivement. l_{fL} retourne le maximum de latence tolérée par une fonction de supervision donnée en paramètre ou *null* si la fonction n'a aucune contrainte de latence. La latence tolérée par une fonction est calculée entre cette fonction et les sondes sources des flux rentrants à cette fonction. Nous considérons que les fonctions utilisateur sont les seules qui ont une latence maximale tolérée car nous ne traitons que la latence de bout en bout. l_{ff} est une fonction d'étiquetage de paires de nœuds qui retourne *vrai* si les fonctions de supervision données en paramètre sont

identiques (*c.-à-d.*, ont le même type et les mêmes paramètres) et retourne $faux$ sinon. Enfin, nous associons une fonction d'étiquetage d'arcs à ce graphe. Il s'agit de l_{oC} qui retourne la capacité requise par le flux sur l'arc.

La construction de l_{ff}, l_{fC} et l_{oC} ne dépend pas des entrées. En effet, l_{ff} peut être déterminée en comparant le type et les paramètres des différentes fonctions. Pour construire l_{fC} et l_{oC}, il faut déterminer les capacités de calcul et de réseau requises par les fonctions de supervision. Cela peut être réalisé à l'aide de modèles théoriques [Chakravarthy et al. 2009] ou de benchmarks. Les autres éléments de G_U (*c.-à-d.*, F, R, l_{fS}, l_{fL}) peuvent être facilement déterminés à partir de la description des besoins des utilisateurs donnée en entrée, car ils sont exprimés par le langage de description présenté dans le chapitre 3.

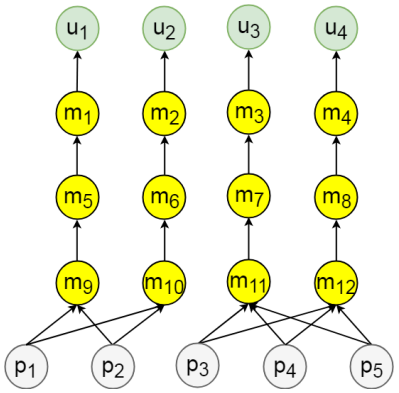

Figure 4.1 – Un exemple de G_U

Modélisation de la description de l'infrastructure

Nous modélisons l'infrastructure par le graphe orienté $G_I = (S, L, l_{sC}, l_{lC}, l_{lL})$. La figure 4.2 représente un exemple de ce graphe. L'ensemble des nœuds S de G_I représente les serveurs de l'infrastructure. L'ensemble de ses arcs L représente les liens réseau reliant les serveurs. Les liens bidirectionnels sont modélisés par deux arcs opposés. Un arc relie chaque nœud à lui-même. Il représente un lien logique destiné à héberger des flux entre les fonctions déployées sur le même serveur. Nous supposons qu'il a une latence nulle et une capacité illimitée. l_{sC} est une fonction d'étiquetage de nœuds. Elle retourne la capacité du serveur qui est représenté par le nœud. l_{lC} et l_{lL} sont des des fonctions d'étiquetage d'arcs. Elles retournent respectivement la capacité et la latence du lien représenté par cet arc.

Les éléments de G_I peuvent être déterminés facilement de la description de l'infrastructure donnée en entrée, car ils sont exprimés par le langage de description présenté dans le Section 3.3.

4.2.3 Définition des variables

Chaque solution de notre problème est une affectation différente de cinq variables.

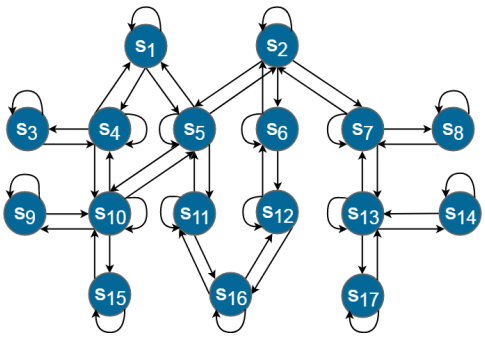

FIGURE 4.2 – An example of G_I

Variables pour les sorties du problème

La résolution du problème de placement mutualisé consiste à déterminer quatre inconnues : les fonctions à instancier, les serveurs qui les hébergent, les flux entre ces fonctions et les liens qui hébergent ces flux. Nous modélisons chacune par une variable.

D'abord, nous définissons les variables V_M et A_M. Elles représentent respectivement les fonctions à instancier et les flux entre elles. Ainsi, $G_M = (V_M, A_M)$ est le graphe qui modélise le service de supervision à instancier. Ensuite, nous définissons $X_{SF} = \{x_{sf_i} | \forall i \in [\![1, |F|]\!]\}$ l'ensemble des variables x_{sf_i} représentant les serveurs qui hébergent les fonctions f_i, $\forall i \in [\![1, |F|]\!]$. Enfin, nous définissons $X_{LO} = \{x_{lo_i} | \forall i \in [\![1, |F \times F|]\!]\}$ l'ensemble des variables x_{lo_i} représentant l'ensemble des liens qui hébergent les flux $o_i \in F \times F$, $\forall i \in [\![1, |F \times F|]\!]$.

Variable pour exprimer les exigences fonctionnelles

Comme l'illustre la figure 4.3, X_{SF} et X_{LO} définissent le couplage entre G_M et G_I. Nous utilisons ces variables pour exprimer les contraintes de couplage de ces graphes.

Afin de simplifier l'expression des contraintes de couplage entre G_U et G_M, nous introduisons $X_{OR} = \{x_{ou_i} | \forall e \in [\![1, |R|]\!]\}$ l'ensemble des variables x_{or_i} représentant les flux de G_M qui sont fonctionnellement équivalents aux flux r_i de G_U, $\forall i \in [\![1, |R|]\!]$.

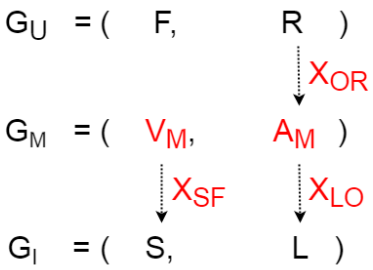

FIGURE 4.3 – Positionnement des variables du problème par rapport à ses entrées

4.2.4 Définition des domaines

Soient D_{V_M}, D_{A_M}, $D_{X_{SF}}$ et $D_{X_{LO}}$ et $D_{X_{OR}}$ les domaines de V_M, A_M, X_{SF} et X_{LO} et X_{OR} respectivement.

Définition de D_{V_M} et D_{A_M}

G_M est construit à partir de G_U car le service de supervision est instancié en fonction des besoins des utilisateurs. La figure 4.4 illustre un exemple de G_M construit à partir de G_U représenté dans la figure 4.1. Les différences entre les deux sont surlignées en rouge : les nœuds et les arcs qui ont été supprimés sont représentés en pointillés et les arcs qui ont été rajoutés sont représentés en ligne continue. Ces différences sont dues à la mutualisation des fonctions équivalentes entre les différents utilisateurs. Dans cet exemple, les fonctions m_9 et m_{10} sont mutualisées entre u_1 et u_2. De plus, m_{11} et m_{12} sont mutualisées entre u_3 et u_4. Enfin, m_7 et m_8 sont mutualisées entre u_3 et u_4.

La mutualisation ne rajoute pas de nœuds à G_M qui ne sont pas dans G_U. Ainsi, l'ensemble de nœuds de G_M est inclus dans celui de G_U (*c.-à-d.*, $V_M \subseteq F$). Par conséquent, $D_{V_M} \subseteq F$. Cependant, la mutualisation peut rajouter des arcs à G_M qui ne sont pas dans G_U. Ainsi, l'ensemble d'arcs de G_M n'est pas inclus dans celui de G_U (*c.-à-d.*, $A_M \not\subseteq R$). Puisque $A_M \subseteq V_M \times V_M$ et $V_M \subseteq F$, alors $A_M \subseteq F \times F$. Par conséquent, $D_{A_M} \subseteq F \times F$.

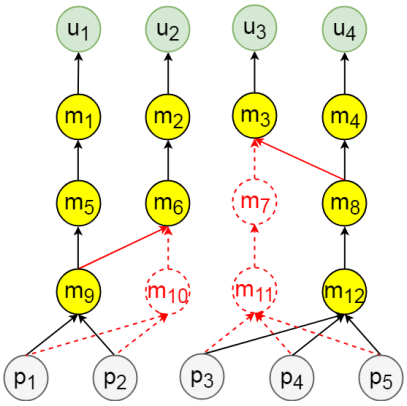

FIGURE 4.4 – Un G_M dérivé de G_U

Définition de $D_{X_{SF}}$ et $D_{X_{LO}}$

$D_{X_{SF}}$ est l'ensemble des serveurs pouvant héberger les fonctions de supervision. Donc, $D_{X_{SF}} = S$.

$D_{X_{LO}}$ est l'ensemble des liens qui peuvent héberger les flux entre les fonctions de supervision. Pour héberger un flux, les liens doivent former un chemin. Ainsi, $D_{X_{LO}} = chemins(G_I)$.

Définition de $D_{X_{OR}}$

$D_{X_{OR}}$ est l'ensemble des flux de G_M qui sont équivalents aux flux de G_U. Cet ensemble est représenté par la variable A_M. Ainsi, $D_{X_{OR}} = D_{A_M} \subseteq F \times F$.

4.2.5 Contraintes du problème

Nous définissons trois types de contraintes pour vérifier : la satisfaction des exigences fonctionnelles des utilisateurs, la satisfaction des exigences de QoS des utilisateurs et les dépendances entre les variables.

Contraintes fonctionnelles

Nous définissons trois contraintes fonctionnelles. Elles établissent le couplage entre G_U et G_M. En premier lieu, chaque flux requis ($r \in R$) qui se dirige vers une fonction de type utilisateur ($tete(r) \in U$) doit avoir un flux équivalent (x_{or}) qui se dirige vers la même fonction $tete(x_{or}) = tete(r)$.

$$\forall r \in R, tete(r) \notin U \vee tete(x_{or}) = tete(r) \tag{4.1}$$

En second lieu, chaque flux requis ($r \in R$) doit avoir un flux équivalent (x_{or}) qui résulte d'une fonction source équivalente.

$$\forall r \in R, l_{ff}(queue(r), queue(x_{or})) = vrai \tag{4.2}$$

Enfin, chaque flux requis ($r \in R$) doit avoir un flux équivalent (x_{or}) qui résulte de la même chaîne de fonctions de traitement. Autrement dit, tout flux requis ($r \in R$) ayant un flux rentrant ($r' \in R$ tq $tete(r') = queue(r)$) doit avoir un flux équivalent (x_{or}) ayant un flux rentrant qui est équivalent à celui de r ($tete(x_{or'}) = queue(x_{or})$). La réciproque doit être vérifiée également. Ainsi, chaque flux de l'ensemble des flux équivalents ($x_{or} \in X_{OR}$) ayant un flux rentrant ($x_{or'} \in X_{OR}$ tq $tete(x_{or'}) = queue(x_{or})$) doit être équivalent à un flux requis ($r \in R$) ayant un flux rentrant qui est équivalent à celui de x_{or} ($tete(r') = queue(r)$).

$$\forall r \in R, \forall r' \in R, tete(r') \neq queue(r) \vee tete(x_{or'}) = queue(x_{or}) \tag{4.3}$$

Contraintes de QoS

Pour vérifier la satisfaction des exigences de QoS, nous définissons des contraintes qui établissent le couplage entre G_M et G_I. Elles peuvent être divisées en deux types de contraintes.

Les contraintes du premier type qui vérifient que l'architecture physique du service de supervision est adaptée à la topologie de l'infrastructure sont au nombre de trois. Premièrement, chaque flux du service de supervision ($o \in A_M$) doit avoir des liens hôtes ($x_{lo} \neq \varnothing$).

$$\forall o \in A_M, x_{lo} \neq \varnothing \tag{4.4}$$

Deuxièmement, chaque fonction du service de supervision $f \in V_M$ qui possède un serveur d'hébergement prédéfini doit être hébergée sur ce serveur ($x_{sf} = l_{fS}(f)$) (les utilisateurs et les sondes sont les fonctions les plus concernées par cette contrainte car elles peuvent avoir des contraintes d'hébergement à proximité des utilisateurs ou des ressources observées, respectivement).

$$\forall f \in V_M, l_{fS}(f) = null \vee x_{sf} = l_{fS}(f) \tag{4.5}$$

Enfin, un flux entrant (sortant, respectivement) à une fonction ($tete(o_1) = f$) doit être hébergé sur des liens dont la destination (source, respectivement) est le serveur hôte de cette fonction ($dest(x_{lo_1}) = x_{sf}$).

$$\begin{aligned}\forall x_{sf} \in X_{SF}, (\forall o_1 \in F \times F, tete(o_1) \neq f \vee x_{sf} = dest(x_{lo_1})) \\ \wedge (\forall o_2 \in F \times F, queue(o_2) \neq f \vee x_{sf} = source(x_{lo_2}))\end{aligned} \tag{4.6}$$

Le deuxième type de contraintes de qualité de service concerne les performances. Nous en définissons trois. Premièrement, chaque serveur de l'infrastructure ($s \in S$) doit avoir plus de capacité ($l_{sC}(s)$) que la somme des capacités requises par les fonctions qu'il héberge ($\sum_{\substack{f \in V_M \\ x_{sf}=s}} l_{fC}(f)$).

$$\forall s \in S, \sum_{\substack{f \in V_M \\ x_{sf}=s}} l_{fC}(f) \leq l_{sC}(s) \tag{4.7}$$

Deuxièmement, chaque lien $l \in L$ de l'infrastructure doit avoir plus de capacité ($l_{lC}(l)$) que la somme des capacités requises par le flux de service de supervision qu'il héberge. La capacité requise par un flux $o \in A_M$ est le maximum de capacités requises par ses flux équivalents ($\max_{\substack{\forall r \in R \\ o \in x_{or}}} l_{oC}(r)$).

$$\forall l \in L, \sum_{\substack{\forall o \in A_M \\ l \in x_{lo}}} (\max_{\substack{\forall r \in R \\ o \in x_{or}}} l_{oC}(r)) \leq l_{lC}(l) \tag{4.8}$$

Enfin, la latence requise par les utilisateurs doit être satisfaite. Ainsi, les liens hébergeant des flux qui forment un chemin vers un utilisateur ($p \in chemins(G_U)$ tq $dest(p) \in Uetr \in p$) doivent avoir une latence inférieure à celle que cet utilisateur tolère ($\sum_{l \in x_{lx_{or}}} l_{lL}(l) \leq l_{fL}(dest(p))$).

$$\forall p \in chemins(G_U), \forall r \in p, dest(p) \notin U \vee \sum_{l \in x_{lx_{or}}} l_{lL}(l) \leq l_{fL}(dest(p)) \tag{4.9}$$

Contraintes de chaînage

Enfin, nous définissons deux contraintes de chaînage qui vérifient des dépendances entre les variables. Premièrement, les fonctions queues et têtes des arcs du graphe du service de supervision (A_M) doivent faire partie de l'ensemble de ses nœuds. En plus, chaque nœud du graphe du service de supervision doit avoir au moins un arc qui le connecte à

un autre nœud de ce graphe. Ainsi, l'ensemble des têtes et queues des arcs du graphe de supervision ($\bigcup_{o \in A_M} \{tete(o), queue(o)\}$) doit être égal à l'ensemble de ses nœuds (V_M).

$$V_M = \bigcup_{o \in A_M} \{tete(o), queue(o)\} \tag{4.10}$$

Deuxièmement, comme il représente un flux du service de supervision, chaque x_{or} doit être dans A_M (*c.-à-d.*, $X_{OR} \subseteq A_M$). Réciproquement, chaque flux du service de supervision doit être équivalent à au moins un flux requis par les utilisateurs (*c.-à-d.*, $A_M \subseteq X_{OR}$).

$$X_{OR} = A_M \tag{4.11}$$

4.2.6 Fonction objectif

La mutualisation du traitement entre différents utilisateurs réduit l'empreinte de calcul. Cependant, elle peut augmenter l'empreinte réseau. En effet, le traitement mutualisé peut nécessiter d'être placé loin d'une partie des utilisateurs pour satisfaire la contrainte de latence d'autres. Pour cette raison, en tant que fonction objectif, nous ne considérons pas la maximisation de la mutualisation mais nous considérons plutôt la minimisation de l'empreinte globale du calcul et du réseau. Nous évaluons la somme des deux empreintes par la variable X_{fp} à minimiser. Nous considérons que les deux empreintes ont le même coût, mais selon le contexte, différents coefficients peuvent être utilisés dans cette somme.

$$X_{fp} = \sum_{\substack{s \in S \\ f \in V_M \\ x_{sf} = s}} l_{sC}(f) + \sum_{\substack{l \in L \\ o \in A_M \\ l \in x_{lo}}} \left(\max_{\substack{\forall r \in R \\ o \in x_{or}}} l_{oC}(r) \right) \tag{4.12}$$

4.3 Évaluation

Pour évaluer la pertinence de notre approche, nous comparons le placement mutualisé (M) avec un placement non mutualisé (NM) par rapport à leurs empreintes sur l'infrastructure et le temps de calcul qu'ils nécessitent.

Pour calculer le NM, nous avons modifié le modèle en supprimant le couplage entre G_U et G_M et en considérant que $G_M = G_U$. Ainsi, les variables V_M, A_M et $x_{or_i} | \ i \in [\![1; |R|]\!]$ sont remplacées par F, R et $r_i | \ i \in [\![1; |R|]\!]$, respectivement. En plus, les contraintes 4.1, 4.2, 4.3, 4.10 et 4.11 liées à ces variables sont supprimées également. Nous gardons la même fonction objectif. Ainsi, l'utilisation des ressources est toujours minimisée mais sans mutualisation des fonctions ni des flux.

Dans le calcul des deux placements, nous nous appuyons sur une stratégie pour explorer efficacement l'espace de recherche. C'est une combinaison des deux stratégies "dernier conflit" [Lecoutre et al. 2009] et "plus petit domaine en premier". Elle consiste à prioriser la sélection des variables impliquées dans le dernier conflit afin d'affecter de nouvelles valeurs à ces variables. S'il n'y a pas de conflit, les variables ayant les plus petits domaines sont sélectionnées. Nous avons implémenté les deux modèles de placement

en utilisant le solveur de contraintes Choco [Jussien et al. 2008] 4.0.6. Le code source est disponible en ligne : https://github.com/edgeMonitoring/PlacementCalculator. Nous avons effectué tous les tests sur une machine ayant un processeur Xeon E5-2640V4 et 64 Go de RAM en utilisant le système d'exploitation Ubuntu 16.04 et une machine virtuelle Java 1.8.0.

4.3.1 Scénarios du test

Nous considérons deux catégories de tests pour analyser séparément l'impact de la modification de chaque entrée du problème. Dans les deux tests UR1 et UR2, nous analysons l'impact de la modification des besoins des utilisateurs et dans les deux tests I1 et I2, nous analysons l'impact de la modification de l'infrastructure.

En ce qui concerne l'impact des besoins des utilisateurs, les tests UR1 et UR2 considèrent des besoins basiques G_U, où tous les utilisateurs doivent effectuer le même nombre de fonctions de traitement sur une sonde commune. Dans le test UR1, nous varions le nombre de fonctions de traitement requises par utilisateur ($|M|/|U|$) tout en conservant le nombre d'utilisateurs fixé à 4 ($|U| = 4$) comme l'illustre la figure 4.5. Inversement, dans le test UR2, nous modifions le nombre d'utilisateurs (| U |) en gardant le nombre de fonctions de traitement requises par utilisateur fixé à 3 ($|M|/|U| = 3$) comme l'illustre la figure 4.6. Nous supposons que chaque fonction nécessite 1 Mo de RAM et que chaque communication entre les fonctions nécessite 1 Mbps de bande passante (ces valeurs ont été choisies à titre d'illustration). Nous générons les types et les paramètres de fonctions de manière aléatoire qui vérifie trois hypothèses. Premièrement, le serveur hôte prédéfini d'une fonction sonde est une ressource à l'Edge (lorsqu'une telle ressource existe dans l'infrastructure). Deuxièmement, entre les serveurs hôtes prédéfinis de chaque fonction utilisateur et ses sondes, il existe au moins un chemin dont la latence est inférieure ou égale au maximum toléré par cet utilisateur. Enfin, 60% des fonctions de traitement ayant le même rang dans les différentes chaînes de traitement sont identiques (les flux sortants de ces fonctions satisfont la contrainte 4.2). À noter que cette hypothèse ne détermine pas le taux de mutualisation attendu car elle ne garantit pas la satisfaction des autres contraintes fonctionnelles (*c.-à-d.*, Contraintes 4.1 et 4.3). Pour les deux tests, l'infrastructure est composée de quatre arbres de serveurs et elle a la configuration des infrastructures du test I1 qui sera détaillée dans le paragraphe suivant.

En ce qui concerne l'impact de l'infrastructure, les tests I1 et I2 reposent sur deux topologies différentes d'infrastructures. Dans le test I1, la topologie est un anneau d'arbres. Chaque arbre est composé de 7 serveurs qui représentent des sites différents : un point de présence central (c_i), deux points de présence régionaux (r_i) et quatre ressources à l'Edge (e_i) tels que chaque c_i est connecté à deux r_i et chaque r_i est connecté à deux e_i (nous modélisons chaque point de présence par un serveur de haute performance car la latence à l'intérieur d'un centre de données est inférieure à 1 ms [Rumble et al. 2011]. Dans ce test, nous varions le nombre d'arbres (| S | / 7), comme l'illustre la figure 4.7. Dans le test I2, nous considérons une topologie partiellement connectée. Nous prenons comme exemple une partie de la topologie de RENATER, le NREN français [Schafer 2015]. Il se compose de huit points de présence centraux c_1 .. c_8, situés respectivement

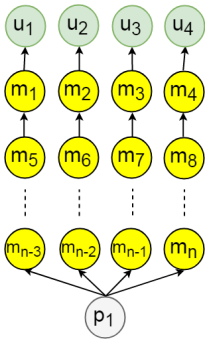

FIGURE 4.5 – Entrées du test UR1

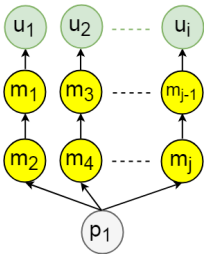

FIGURE 4.6 – Entrées du test UR2

à Rennes, Paris, Strasbourg, Bordeaux, Lyon, Toulouse, Marseille et Nice, ainsi qu'un point de présence régional r situé à Orsay. La figure 4.7 décrit cette infrastructure. Pour le test I1, nous supposons que les latences des liens interconnectant les différents c_i, connectant un c_i avec un r_i et connectant un r_i avec un e_i sont 22 ms, 12 ms et 8 ms, respectivement (Ces valeurs ont été choisies à titre d'illustration). Pour le test I2, nous considérons les latences de liens fournies par RENATER [RENATER 2007]. Pour les deux types d'infrastructures, nous supposons que les capacités d'hébergement des c_i, r_i et e_i sont de 10 Go, 1 Go et 100 Mo, respectivement. De plus, nous considérons que les bandes passantes des liens interconnectant les c_i, connectant un c_i avec un r_i et connectant un r_i avec un e_i sont de 10 Gbps, 1 Gbps et 100 Mbps, respectivement. Ces valeurs des capacités d'hébergement ont été choisies pour garantir l'existence d'un NM. Sinon, il est possible de se trouver dans le cas où il y a des solutions pour M et pas de solutions pour NM. Pour simplifier l'étude, nous ne considérons pas ce cas. Dans les deux tests I1 et I2, nous supposons qu'il n'y a pas de restrictions de routage (c'est la configuration la plus difficile car elle maximise le nombre de chemins réseau). Enfin, l'entrée des besoins des utilisateurs est composée de 4 utilisateurs ($|U| = 4$) nécessitant chacun d'effectuer 3 fonctions de traitement ($|M|/|U| = 3$) sur la même sonde. Les capacités de calcul et de réseau dont ils ont besoin ont été définies conformément aux tests UR1 et UR2.

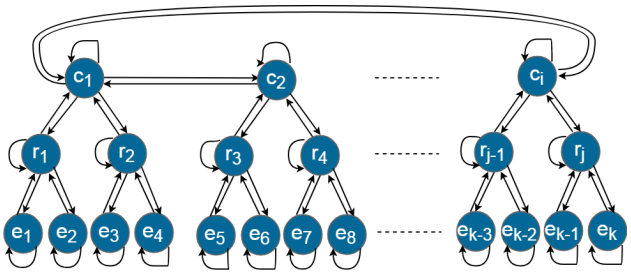

FIGURE 4.7 – Entrées du test I1

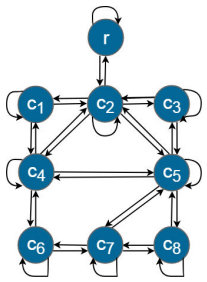

FIGURE 4.8 – Entrées du test I2

4.3.2 Résultats du test

Dans cette section, nous comparons le M avec le NM par rapport au temps de calcul requis ainsi que leur empreinte de calcul et réseau. Nous considérons les premières et dernières solutions trouvées en 10 minutes d'exécution. Les résultats sont résumés dans la table 4.2 et la table 4.3. Chaque valeur dans ces tables est la moyenne des résultats de 30 expériences qui diffèrent au niveau des paramètres générés aléatoirement. Les quartiles sont affichés sur les figures 4.9, 4.10, 4.11 et 4.12. Le gain d'empreinte réalisé par le M est représenté sous la forme d'un pourcentage de l'empreinte de NM. Dans 3% des expériences du test I1 où le nombre d'arbres est égal à 2 ($|S|/7 = 2$), l'espace de recherche a été exploré entièrement lors du calcul du NM. Néanmoins, 10 minutes n'étaient pas suffisantes pour trouver une solution M dans 13% des expériences du test UR1 où $|M|/|U| = 6$ et dans 27 % des expériences du test UR1 où $|M|/|U| = 8$.

Test	**Entrées**				**Temps (s)**		**Emp. calcul (MB)**		**Emp. réseau (Mbps)**	
	|M|/|U|	**|U|**	**|F|**	**|S|/7**	**NM**	**M**	**NM**	**M**	**NM**	**M**
UR1	2	4	13	4	0.034	1.878	13	10 (-23%)	24	19 (-21%)
	4	4	21	4	0.085	27.935	21	17 (-19%)	32	26 (-19%)
	6	4	29	4	0.153	56.985	29	25 (-14%)	40	34 (-15%)
	8	4	37	4	0.187	63.16	37	32 (-14%)	48	42 (-13%)
UR2	3	3	13	4	0.046	1.542	13	12 (-8%)	21	19 (-10%)
	3	5	21	4	0.078	3.988	21	18 (-14%)	35	30 (-14%)
	3	7	29	4	0.106	16.527	29	21 (-28%)	50	38 (-24%)
	3	9	37	4	0.148	32.98	37	28 (-24%)	64	50 (-22%)
I1	3	4	17	2	0.007	0.276	17	13 (-24%)	26	19 (-27%)
	3	4	17	4	0.078	6.578	17	13 (-24%)	28	22 (-21%)
	3	4	17	6	0.191	29.475	17	13 (-24%)	31	25 (-19%)
	3	4	17	8	0.405	99.301	17	13 (-24%)	33	26 (-21%)
I2	3	4	17	9/7	0.008	0.358	17	13 (-24%)	19	15 (-21%)

TABLE 4.2 – Comparaison entre les premières solutions du NM et celles du M

Test	**Entrées**				**Temps (s)**		**Emp. calcul (MB)**		**Emp. réseau (Mbps)**	
	|M|/|U|	**|U|**	**|F|**	**|S|/7**	**NM**	**M**	**NM**	**M**	**NM**	**M**
UR1	2	4	13	4	223.298	59.095	13	10 (-23%)	20	16 (-20%)
	4	4	21	4	186.944	157.614	21	17 (-19%)	29	24 (-17%)
	6	4	29	4	119.902	166.238	29	25 (-14%)	38	32 (-16%)
	8	4	37	4	88.282	102.676	37	32 (-14%)	46	41 (-11%)
UR2	3	3	13	4	232.378	252.853	13	12 (-8%)	17	15 (-12%)
	3	5	21	4	218.736	173.373	21	18 (-14%)	31	27 (-13%)
	3	7	29	4	176.994	192.797	29	21 (-28%)	46	35 (-24%)
	3	9	37	4	116.676	93.053	37	28 (-24%)	60	49 (-18%)
I1	3	4	17	2	277.179	63.126	17	13 (-24%)	19	15 (-21%)
	3	4	17	4	231.245	153.024	17	13 (-24%)	24	19 (-21%)
	3	4	17	6	120.906	221.019	17	13 (-24%)	28	22 (-21%)
	3	4	17	8	82.793	227.968	17	13 (-24%)	30	24 (-20%)
I2	3	4	17	9/7	95.717	20.37	17	13 (-24%)	17	13 (-24%)

TABLE 4.3 – Comparaison entre les dernières solutions du NM et celles du M

Analyse de l'empreinte

Les résultats montrent que l'empreinte de calcul de NM est la même dans les premières et les dernières solutions. Elle est égale à la somme des capacités de calcul requises par les fonctions (*c.-à-d.*, $|F| \times 1MB$) car ces dernières n'ont pas été mutualisées. L'empreinte de calcul de M est également la même dans les premières et dernières solutions, mais elle est inférieure à la somme des capacités de calcul requises par les fonctions. Ainsi, un G_M est trouvé. Cependant, il est le seul car le calcul de ses placements possibles nécessite plus de 10 minutes, comme dans la plupart des cas pour NM où 10 minutes n'ont pas été suffisantes pour parcourir intégralement l'espace de recherche.

Dans les différentes expériences, pour les deux placements M et NM, les premières et les dernières solutions trouvées ont le même nombre de fonctions. Elles ont alors le même nombre de flux. Cependant, les dernières solutions ont des empreintes réseau inférieures aux premières. Cela est dû au fait que le solveur parvient à placer les flux sur un nombre inférieur de liens. Le test I1 montre comment le nombre de liens connectant les serveurs affecte l'empreinte réseau car, pour les mêmes besoins des utilisateurs, l'empreinte du réseau augmente lorsque la taille de l'infrastructure augmente.

Comme attendu, la figure 4.9 montre que la première et la dernière solution de M ont moins d'empreinte de calcul que celles de NM. Le gain réalisé varie de -8% à -28%. De même, la figure 4.10 montre que la dernière solution de M a moins d'empreinte réseau que celle de NM. Le gain réalisé varie de -11% à -24%. La figure 4.11 montre que, dans la plupart des cas, la première solution de M a même une empreinte réseau qui est inférieure à la dernière solution trouvée pour NM (ce qui est toujours le cas pour l'empreinte de calcul). Cela est dû au fait que les solutions M ne peuvent pas être atteintes par le calcul de solutions de NM.

Analyse du temps de calcul

La taille des entrées affecte significativement le temps de calcul. La table 4.2 montre que le temps nécessaire pour trouver les premières solutions augmente d'une manière exponentielle lorsque la taille des entrées augmente linéairement. Cela est dû au fait que le calcul de M et NM est un problème NP-complet.

Pour les différents tests, la figure 4.12 montre que la première solution du M nécessite plus de temps que celle du NM. De plus, comme détaillé précédemment, 10 minutes ne sont pas suffisantes pour trouver une solution dans certaines expériences pour le calcul de M. Cependant, elles sont suffisantes dans d'autres expériences pour explorer entièrement l'espace de recherche dans le calcul de NM. Cela est dû au fait que la complexité du problème M est supérieure à celle de NM. Le premier consiste en deux problèmes NP-complets (*c.-à-d.*, contraction d'arêtes et couplage de graphes) alors que le second ne comprend qu'un seul (*c.-à-d.*, couplage de graphes). Étant donné que toute solution pour le NM est viable pour le M, il est possible de commencer le calcul de M en recherchant un NM. Cela revient à ajouter cette contrainte : $\forall r \in R, head(x_{ou}) = head(r) \wedge tail(x_{ou}) = tail(r)$ et à la relâcher une fois une solution a été trouvée. Ainsi, il est possible de réduire le temps de calcul du M.

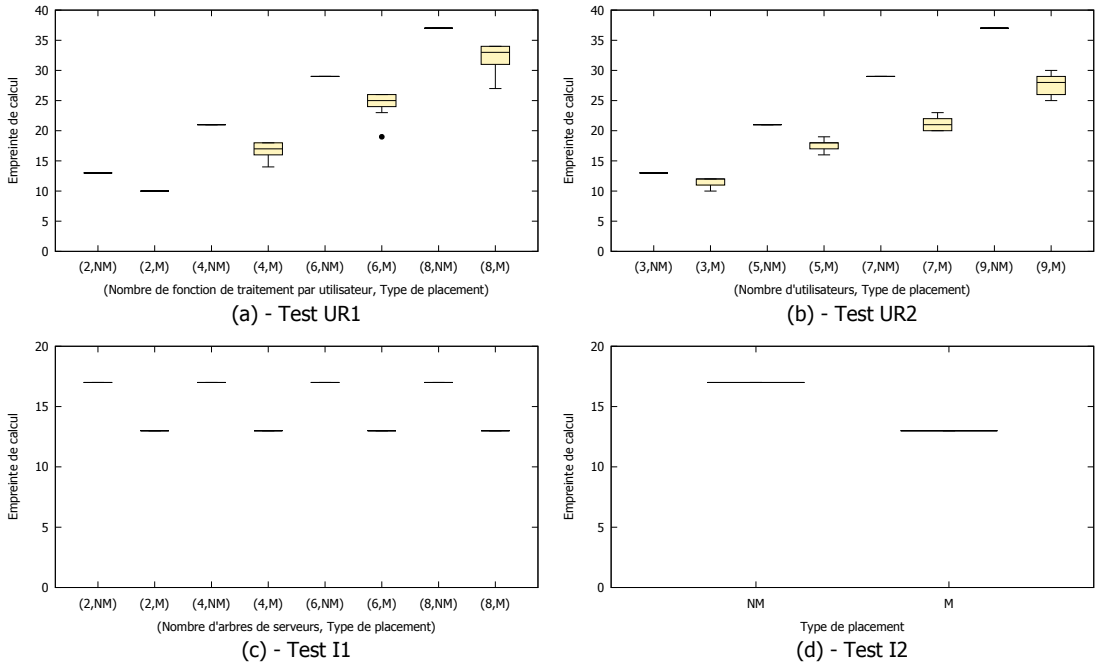

FIGURE 4.9 – Empreinte de calcul de la première et la dernière solution pour M et NM

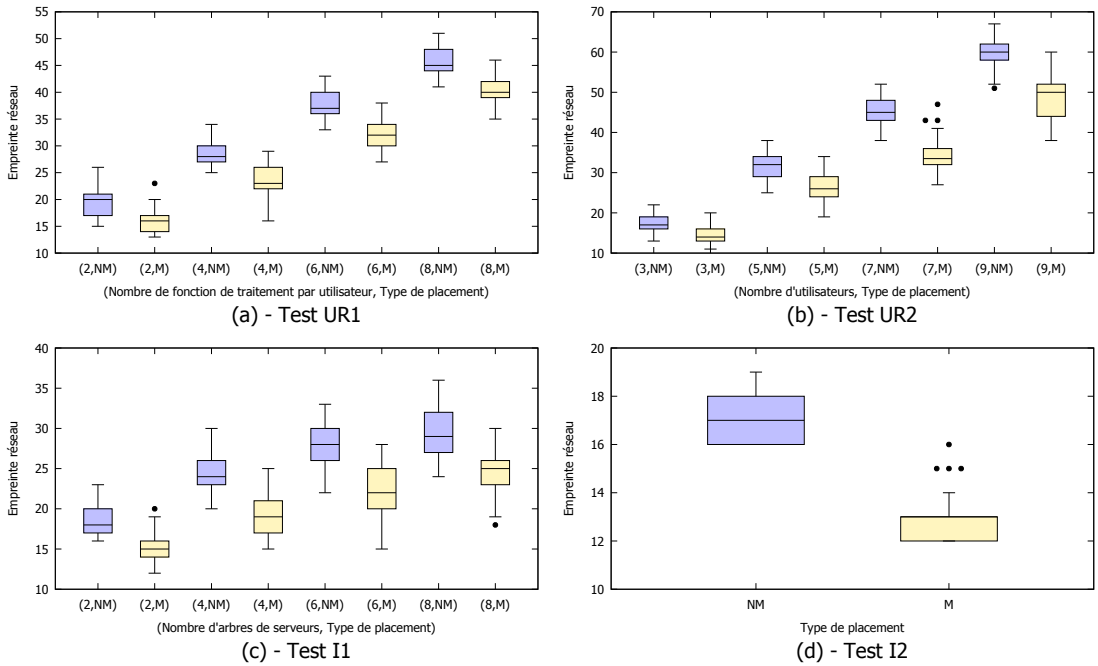

FIGURE 4.10 – Empreinte réseau des dernières solutions pour M et NM

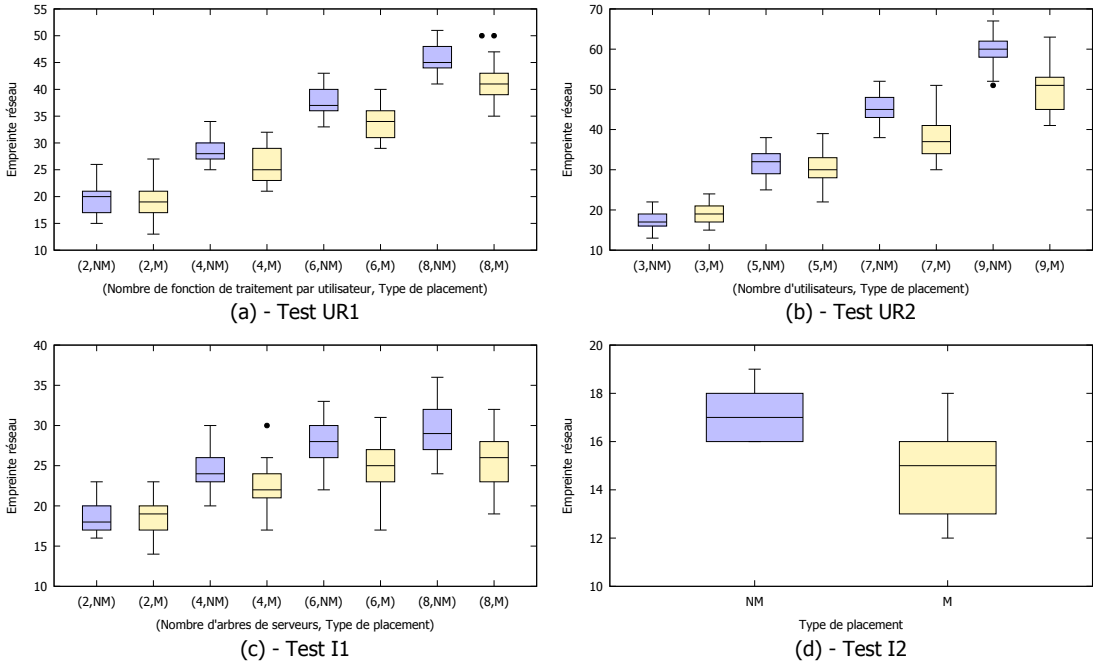

FIGURE 4.11 – Empreinte réseau des dernières solutions de N et des premières solutions de NM

Pour le même nombre de fonctions (*c.-à-d.*, |F|), la table 4.2 montre qu'il est plus difficile pour les deux placements de trouver une première solution dans les tests ayant le plus grand nombre de fonctions de traitement. Cela est dû au fait que ces fonctions introduisent plus d'inconnues au problème que les autres fonctions (*c.-à-d.*, utilisateurs et sondes). En effet, elles peuvent être mutualisées ou pas. De plus, leurs serveurs hôtes ne sont pas prédéfinis.

Par rapport à une infrastructure de test I1 composée de deux arbres ($|S| = 14$, $|L| = 40$), l'infrastructure de test I2 a un nombre de serveurs et un nombre de liens plus réduits ($|S| = 9$, $|L| = 35$). Cependant, la figure 4.12 montre que pour les deux placements, il est plus difficile de trouver la première solution dans le dernier test que dans le premier. Cela est dû à la différence des topologies d'infrastructures, ce qui se traduit par des nombres de chemins réseau différents. En effet, pour l'infrastructure du test I1 composée de 2 arbres, il n'y a qu'un seul chemin entre deux paires de serveurs. Cependant, pour l'infrastructure de test I2, il existe au moins deux chemins entre toute paire de serveurs, à l'exception de c_2 et r, qui sont connectés par un seul chemin.

4.4 Conclusion

Dans ce chapitre, nous avons formalisé le problème de placement mutualisé. Nous avons commencé par présenter le problème en établissant le lien avec les travaux existants. Le concept de la mutualisation n'apparaît que partiellement dans les études qui traitent

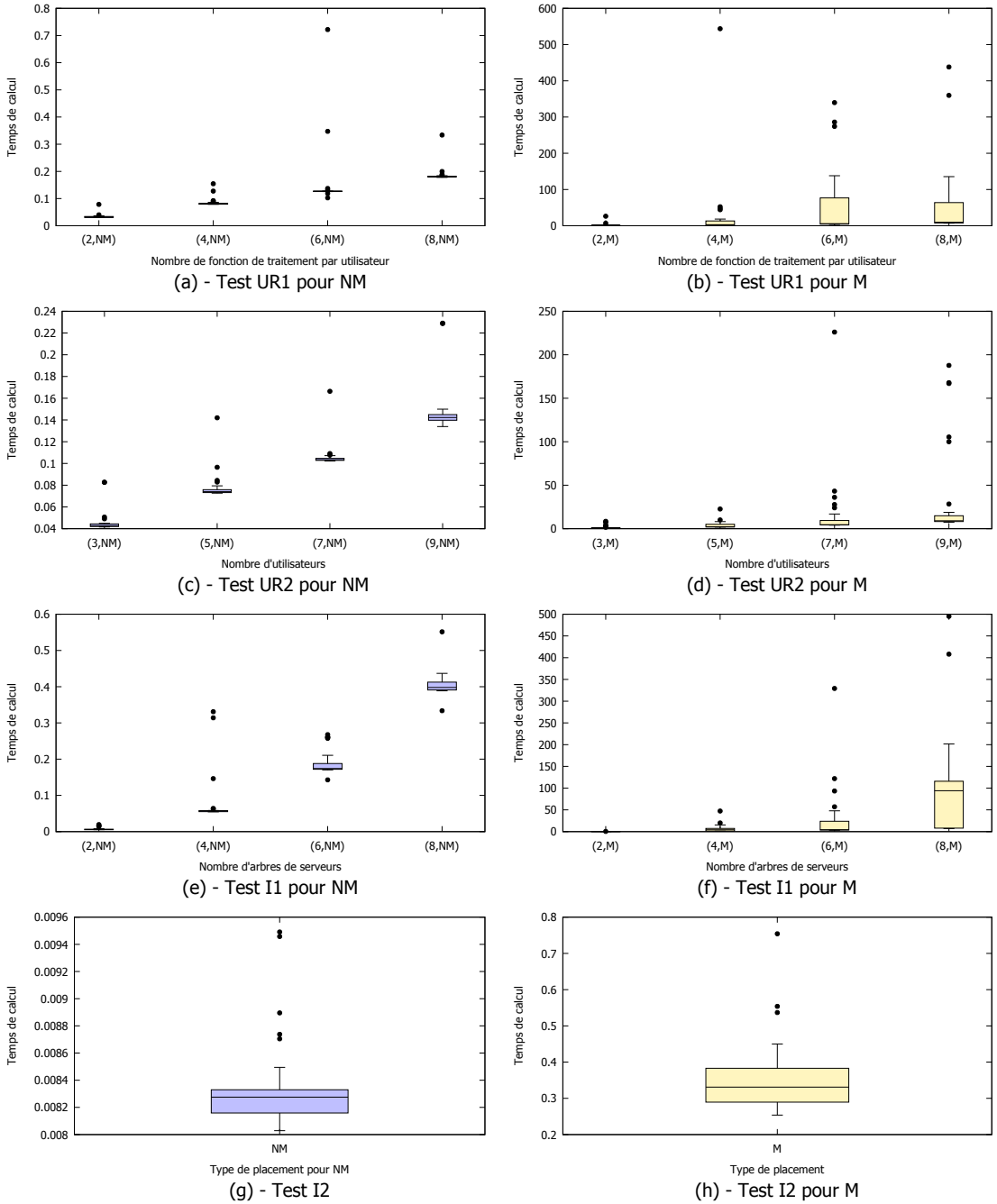

FIGURE 4.12 – Temps de calcul nécessaire pour trouver une première solution de M et NM

le problème de chaînage de fonctions réseau. Dans ce contexte, il a été proposé de dédier la même fonction réseau à des utilisateurs différents. Notre proposition consiste à pousser la mutualisation davantage en regroupant le traitement et les flux identiques à des utilisateurs différents. Étant donné que le calcul des mutualisations est NP-complet et orienté par les contraintes, nous avons opté pour le modéliser par un problème de satisfaction de contraintes. Dans notre modèle, nous considérons essentiellement deux types de contraintes. Il s'agit des contraintes fonctionnelles et des contraintes de qualité de service requis par les utilisateurs. Comme fonction objectif, nous considérons la minimisation de l'empreinte de calcul et réseau du service de supervision. Les tests que nous avons réalisés ont montré la pertinence de notre approche dans la réduction de l'empreinte de supervision. Cependant, ils ont montré qu'elle augmente le temps de calcul. Pour faire face à cette limite, il serait intéressant de cibler en premier lieu un placement non mutualisé et de poursuivre la recherche d'une solution mutualisée si le SLA défini avec l'utilisateur le permet.

IV

Conclusion générale & perspectives

5

Conclusion générale

Dans cette thèse, nous avons proposé un canevas logiciel pour la mise en place d'un service de supervision holistique qui peut être utilisé par tous les tenants d'une infrastructure Edge. Ce canevas a pour objectif la détermination de l'architecture de déploiement des fonctions de base de supervision : observation, traitement et exposition des mesures. Ces fonctions sont structurées sur la base d'un calcul de placement qui prend en compte les contraintes fonctionnelles et de qualité de service des utilisateurs. Le calcul de ce placement s'appuie sur une description des besoins des utilisateurs ainsi qu'une description des capacités de l'infrastructure. Pour faire face à la diversité de ces besoins et à l'hétérogénéité de l'infrastructure Edge, nous avons utilisé un langage de description pour chacun des deux. En outre, le placement est calculé avec l'objectif de minimiser l'empreinte de calcul et réseau du service de supervision. À cette fin, nous avons proposé de mutualiser le traitement et les flux identiques entre des utilisateurs différents.

Dans le chapitre 1, le contexte de ce travail est introduit. Nous avons commencé par présenter le Cloud Computing. Ce paradigme permet d'offrir les ressources (par exemple, le calcul, stockage et réseau) en tant que services. Les fournisseurs du Cloud ont tendance à s'appuyer sur des infrastructures centralisées et massives afin de réduire le coût de maintenance et de réaliser une économie d'échelle. Cependant, ce type d'infrastructure entrave la satisfaction des exigences de latence et de bande passante des applications de nouvelle génération. Celles-ci requièrent de plus en plus de ressources à l'Edge (par exemple, ordinateur bureautique, passerelle résidentielle, téléphone portable) qui se caractérisent par leur l'hétérogénéité, distribution et dynamicité. Par conséquent, elles rendent la gestion de l'infrastructure et, en particulier, sa supervision plus complexes.

En considérant les caractéristiques de l'infrastructure Edge (*c.-à-d.*, hétérogénéité, distribution et dynamicité), nous avons spécifié, dans le chapitre 2, cinq propriétés à

satisfaire par la solution de supervision. Ces propriétés sont la compatibilité vis-à-vis des ressources hétérogènes, le passage à l'échelle, la tolérance aux pannes réseau, la tolérance aux pannes des serveurs et la proximité vis-à-vis des utilisateurs. Étant donné le nombre et la diversité des solutions de supervision existantes, il est difficile d'étudier et de mettre en œuvre tous les scénarios possibles pour évaluer expérimentalement leur capacité à satisfaire ces propriétés. Nous avons opté, à la place, pour une évaluation qualitative basée sur les architectures des solutions. Nous avons considéré deux aspects architecturaux qui répondent différemment aux propriétés spécifiées. Le premier aspect est la décomposition fonctionnelle. Elle peut être inexistante, élémentaire ou avancée. Le deuxième aspect est le modèle architectural. Il peut être centralisé, hiérarchique ou pair-à-pair. L'architecture qui répond au mieux aux propriétés spécifiées est celle qui a une décomposition fonctionnelle avancée et un modèle architectural pair-à-pair. À notre connaissance, il n'existe pas de solutions de supervision qui répondent à cette architecture. La mise en œuvre d'une telle solution requiert la définition d'une stratégie pour le placement des fonctions élémentaires afin de maîtriser la flexibilité qu'offre cette architecture décentralisée.

Dans le chapitre 3, nous avons présenté notre canevas logiciel pour la supervision de l'infrastructure Edge. En plus des modules qui assurent l'observation, le traitement et l'exposition des mesures, il intègre un autre module qui calcule leur placement en vue de déterminer l'architecture de la solution de supervision à déployer. Ce module a comme objectif la réduction de l'empreinte de calcul et réseau du service de supervision. À cette fin, nous avons proposé de mutualiser les fonctions de traitement et les flux identiques qui sont requis par des utilisateurs différents. Nous avons montré la pertinence de cette approche pour un service de diffusion de contenus à la périphérie du réseau. En contrepartie, nous avons identifié la diversité des besoins des utilisateurs ainsi que l'hétérogénéité, la distribution et la dynamicité de l'infrastructure Edge comme des challenges à la mise en œuvre de notre approche. Nous nous sommes concentrés sur les trois premiers challenges. Pour faire face à la diversité des besoins des utilisateurs et l'hétérogénéité de l'infrastructure Edge, nous avons proposé l'utilisation de langages qui unifient la description de chacun des deux. Pour faire face à la distribution de l'infrastructure, nous avons pris en considération les contraintes de latence des utilisateurs dans le calcul de mutualisation. Le chapitre est terminé par la description de l'architecture du canevas logiciel proposé. Elle illustre le rôle des différents modules : le "gestionnaire du placement" reçoit en entrée l'expression des besoins de supervision des utilisateurs et les capacités de l'infrastructure et calcule le placement des fonctions de supervision. L'"ordonnanceur de placement" déploie les modules de supervision (qui observent, traitent et exposent les mesures) de telle sorte à minimiser le temps nécessaire pour cette opération.

Dans le chapitre 4, nous avons formalisé le problème de placement mutualisé. Nous avons commencé par établir le lien avec les travaux existants comme le chaînage des fonctions réseau. Le concept de la mutualisation n'apparaît que partiellement dans les études qui traitent ce problème. Dans ce contexte, il a été proposé de dédier la même fonction réseau à des utilisateurs différents. Notre proposition consiste à pousser la mutualisation davantage en partageant le traitement et les flux identiques entre

des utilisateurs différents. Étant donné que le calcul de mutualisation est NP-complet et orienté par les contraintes, nous avons choisi de le modéliser par un problème de satisfaction de contraintes. Notre modèle rend en sortie quatre éléments : les fonctions de traitement, leur serveur hôte, les flux entre les fonctions et leurs liens hôtes. Le modèle vérifie la satisfaction de deux types de contraintes : des contraintes fonctionnelles (*c.-à-d.*, les fonctions placées réalisent le traitement demandé par les utilisateurs) et des contraintes de qualité de service (*c.-à-d.*, les fonctions sont placées sur des ressources qui disposent de capacités suffisantes et la latence maximale autorisée par chaque utilisateur est respectée). La fonction objectif du modèle consiste à minimiser l'empreinte de calcul et réseau du service de supervision. Les tests que nous avons réalisés ont montré la pertinence de notre approche dans la réduction de l'empreinte de supervision. Elle a permis de réaliser un gain entre -8% à -28% sur l'empreinte de calcul et un gain entre -10% à -24% sur l'empreinte réseau. Cependant, les tests ont montré que notre approche augmente le temps de calcul. Pour faire face à cette limite, il serait intéressant de cibler en premier lieu un placement non mutualisé et d'itérer par la recherche d'une solution mutualisée suite au déploiement.

Ainsi, ce travail nous a permis d'aborder la supervision qui est un problème majeur pour les opérateurs d'infrastructure. Le canevas logiciel que nous avons proposé permettra à de tels opérateurs de disposer d'un outillage composable et reconfigurable à souhait pour superviser l'infrastructure Edge. Étant donné l'importance des performances attendues par la 5G, nous avons considéré cet aspect lors de la détermination de la composition. Nous avons proposé de mutualiser les fonctions et les flux entre des utilisateurs différents pour réduire l'empreinte de calcul et réseau sur l'infrastructure. Nous avons montré en implémentant deux calculateurs différents de placement des fonctions de supervision, que notre approche réduit significativement l'empreinte de la supervision sur l'infrastructure. En contrepartie, elle augmente considérablement le temps de calcul lorsque la taille des entrées est grande. Ce défi, lié au passage à l'échelle du calcul de placement, est aujourd'hui abordé au travers d'un post-doctorat financé dans le cadre de l'initiative Discovery [1] au sein du laboratoire Inria (collaboration entre les équipes CORSE et STACK). Il est intéressant de souligner également l'implication du principal développeur du solveur de contraintes Choco dans cette étude (les défis au niveau scalabilité qu'il engendre a suscité un vif intérêt par la communauté en recherche opérationnelle de IMT-Atlantique).

Les codes réalisés autour du calculateur sont disponibles sur un dépot github[2]. Ces codes ont été intégrés au projet Plug'in mené par Orange, qui cible la mise en œuvre d'une plate-forme d'expérimentation pour développer et tester les composants logiciels 5G et les chaînes de fonctions de réseaux virtualisées [Bousselmi et al. 2018]. Enfin, ce travail de doctorat a donné l'opportunité d'encadrer un stage d'ingénieur de deux mois en vue de développer l'ensemble des éléments du canevas. Les travaux menés durant ce stage ont permis de concevoir un prototype d'"ordonnanceur de placement" développé en Python.

[1] http ://beyondtheclouds.github.io/ (visité le 01/10/2018)
[2] https ://github.com/edgeMonitoring/PlacementCalculator (visité le 01/10/2018)

Dans le chapitre suivant, nous donnons plusieurs perspectives possibles à notre travail.

6

Perspectives

En perspective, nous identifions cinq axes de recherche afin de poursuivre notre travail. Ils concernent l'évaluation des langages de description définis, le calcul de mutualisation, la mise en œuvre de l'architecture de déploiement, la généralisation de l'approche proposée à l'Internet des Objets et enfin la coopération entre différents gestionnaires de services.

6.1 Évaluation des langages de description définis

Pour faire face à la diversité des besoins des utilisateurs et à l'hétérogénéité de l'infrastructure Edge, nous avons défini deux langages qui unifient leur expression. Ces langages pourraient faire l'objet d'évaluation approfondie.

Différents critères peuvent être pris en considération dans l'évaluation des langages. Kahraman et al. en identifient dix [Kahraman et al. 2015] : utilisabilité (facilité d'utilisation), réutilisabilité (facilité à exploiter les descriptions élaborées à l'aide du langage par d'autres types de modules), convenance fonctionnelle (capacité à couvrir toutes les fonctions du domaine), fiabilité (les règles du langage permettent d'éviter les erreurs), maintenabilité (facilité de rajout de concepts supplémentaires), extensibilité (facilité de rajout d'options supplémentaires), productivité (capacité à réduire les ressources mobilisées afin d'établir la description), compatibilité (compatibilité avec le domaine), expressivité (facilité de l'expression des concepts) et intégrabilité (facilité d'intégation avec d'autres langages). Il est possible de s'appuyer sur des approches empiriques [Freire et al. 2014] ou qualitatives [Kahraman et al. 2015] pour évaluer les langages par rapport aux différents critères.

6.2 Calcul de mutualisation

Le calcul de mutualisation peut être amélioré sur trois plans.

6.2.1 Réduction du temps de calcul

Les tests que nous avons réalisés ont montré que le calcul de mutualisation augmente significativement le temps de calcul lorsque la taille des entrées (*c.-à-d.*, besoins de supervision des utilisateurs et description de l'infrastructure) est grande. Pour que notre approche soit adaptée à la supervision d'une infrastructure Edge, il est primordial de réduire le temps de calcul.

Le temps de calcul considérable est dû à la complexité du calcul qui consiste à résoudre deux problèmes NP-complets. Le premier est un problème de contraction d'arcs. Il vérifie que la mutualisation satisfait les contraintes fonctionnelles. Le deuxième est un problème de couplage de graphes. Il vérifie que la mutualisation satisfait les contraintes de qualité de service. Pour réduire le temps de calcul des mutualisations, il est possible de résoudre ces problèmes d'une manière itérative au lieu de les résoudre simultanément. Ainsi, un placement mutualisé qui vérifie les contraintes fonctionnelles est déterminé dans un premier temps. Ensuite, la capacité de ce placement à vérifier les contraintes de qualité de service est vérifiée dans un second temps. Cette approche permet de paralléliser la vérification de la qualité de service pour des mutualisations différentes qui vérifient les contraintes fonctionnelles. Par conséquent, le temps de calcul peut être réduit considérablement. Néanmoins, la résolution risque de nécessiter plus de capacité de calcul.

6.2.2 Résilience à la dynamicité de l'infrastructure Edge

Comme détaillé dans la section 1.3.3, l'infrastructure Edge se caractérise par sa dynamicité. Par conséquent, sa topologie et les capacités de ses ressources hôtes peuvent changer au cours du calcul des mutualisations. Ainsi, les mutualisations identifiées risquent de ne pas être adaptées.

Pour faire face à ce défi, il est possible de s'appuyer sur des méthodes de résolution dynamique [Verfaillie et al. 2005]. Ces dernières sont destinées à la résolution des problèmes de satisfaction de contraintes dont les variables, domaines ou contraintes changent en fonction du temps. Leur principe consiste à considérer le problème de satisfaction des contraintes dynamiques comme une séquence de problèmes de satisfaction de contraintes statiques. Suite à tout changement dans le problème, un nouveau problème statique est défini comme étant une relaxation ou une restriction du problème précédent. Différentes techniques sont utilisées pour exploiter le calcul réalisé dans la résolution d'un problème pour résoudre le problème suivant. Elles peuvent être classées en des approches réactives ou en des approches pro-actives selon leur capacité à anticiper les changements.

6.2.3 Extension du modèle

Le modèle de placement mutualisé peut être étendu pour considérer d'autres aspects qui peuvent avoir un impact sur le calcul du placement comme la zone de confiance des utilisateurs, la durée de disponibilité des ressources hôtes, l'énergie dont ces ressources disposent ainsi que leur résilience aux pannes. Nous adressons spécifiquement les deux derniers aspects car la mutualisation peux être exploitée pour mieux les prendre en considération.

Réduction de l'empreinte énergétique

Dans le modèle que nous avons proposé, nous nous sommes concentrés sur le gain de calcul et de réseau qui peut être réalisé en mutualisant les fonctions et les flux identiques. En plus de ces deux critères de performance, il est important de considérer l'aspect énergétique. En effet, comme détaillé dans la section 1.3.3, les ressources au plus bas niveau de l'infrastructure Edge peuvent avoir une énergie limitée (par exemple, téléphone portable). Pour exploiter ces ressources efficacement, il faut considérer la consommation énergétique lors du placement [Zhang et al. 2018]. Pour ce faire, il faut rajouter des fonctions d'étiquetage qui retournent l'énergie disponible au niveau des serveurs ainsi que l'énergie requise par les fonctions. De plus, il faut rajouter une contrainte analogue à 4.8 où les fonctions d'étiquetage des capacités sont remplacées par des fonctions d'étiquetage d'énergie. Enfin, il faut considérer la minimisation de l'empreinte énergétique dans la fonction objectif.

Prise en considération de la résilience aux pannes des ressources hôtes

La résilience aux pannes des ressources hôtes représente un critère de performance important dans le placement sur l'infrastructure Edge. En effet, comme détaillé dans la section 1.3.3, le niveau le plus bas de l'infrastructure est constitué d'équipements qui ne sont pas dédiés à l'infrastructure et qui sont particulièrement vulnérables aux déconnexions. Pour cela, des travaux récents ont considéré la résilience des ressources hôtes dans le placement [Spinnewyn et al. 2017].

Dans le modèle que nous avons proposé, nous n'avons pas considéré cet aspect. Pour en tenir compte, nous n'envisageons pas de nous contenter de la rajouter en tant que dimension supplémentaire de placement (*c.-à-d.*, considérer qu'il y a des résiliences requises par les fonctions et les flux ainsi que des résiliences offertes par les serveurs et les liens). Nous envisageons plutôt de tirer profit des fonctions identiques qui ne sont pas mutualisées à cause des contraintes de latence. En effet, de telles fonctions représentent des pairs qui peuvent se substituer momentanément les uns aux autres si l'un d'eux est en panne ou est inaccessible. Ainsi, cette propriété peut améliorer la résilience du placement.

La mise en œuvre de cette approche nécessite d'étendre notre modèle. Il faut définir pour chaque flux x_{or_i} placé sur l'infrastructure les flux qui peuvent le remplacer lorsque celui-ci n'est pas fonctionnel. Pour cela, il est possible de définir pour chaque flux r_i requis par un utilisateur l'ensemble de flux identiques x_{rr_i} qui sont requis par d'autres

utilisateurs. Pour assurer leur concordance, il faut vérifier que ces flux soient sortants de fonctions identiques et que ces fonctions reçoivent en entrées des flux identiques. Cela se ramène à rajouter deux contraintes analogues à 4.2 et 4.3 avec la substitution de la variable x_{or_i} par x_{rr_i}. Par la suite, le flux qui peut remplacer x_{or_i} est $x_{ox_{rr_i}}$. Pour vérifier que les résiliences requises par les fonctions et les flux sont respectées, il faut rajouter deux contraintes analogues à 4.8 et 4.7 où les fonctions d'étiquetage de capacité sont substituées par des fonctions d'étiquetage de résilience.

6.3 Mise en œuvre de l'architecture de déploiement

Pour donner plus d'interopérabilité aux éléments qui constituent le canevas logiciel, il serait intéressant de décrire, à l'aide d'un langage standardisé, l'architecture de déploiement qui est déduite du calcul de placement. Ainsi, l'utilisateur n'est pas contraint à utiliser la brique de déploiement du canevas que nous proposons. Il peut, à la place, utiliser toutes les solutions d'orchestration et de déploiement qui supportent ce langage.

TOSCA [Verfaillie et al. 2005] est l'un des langages les plus utilisés à cette fin. Il a été standardisé par l'organisation pour l'évolution des normes d'information structurée (OASIS) [1]. Il permet de décrire les applications destinées à être hébergées sur le Cloud. En plus de son moteur de déploiement natif OpenTOSCA [Binz et al. 2013], ce langage est supporté par plusieurs autres moteurs de déploiement comme Cloudify [2] et Ubicity [3].

Pour pouvoir déployer une application décrite avec un langage comme TOSCA, il faut que les briques logicielles de l'application soient construites en avance. Pour les construire, il est possible de concevoir un module qui permet, à partir de l'architecture de déploiement, de générer le code source des briques logicielles et de le compiler. Une autre approche possible est de décrire l'architecture de déploiement à l'aide d'un outil comme Calvin [Persson et al. 2015]. Cet outil est destiné à un déploiement sur les infrastructures Cloud et IdO. Il se charge de la construction des briques logicielles, leur déploiement et la gestion de leur cycle de vie.

6.4 Application de la mutualisation dans le contexte de l'IdO

Dans cette thèse, nous nous sommes concentrés sur la supervision de l'infrastructure Edge. Pour cela, nous avons proposé de mutualiser le traitement et les flux de mesures. Cependant, cette proposition peut être appliquée dans un contexte plus large qui est l'IdO. En effet, les objets connectés peuvent être partagés par plusieurs utilisateurs. Par exemple, les observations remontées par une caméra de vidéo surveillance peuvent intéresser à la fois un bureau de transport et un bureau de police. Au lieu d'effectuer un traitement spécifique pour chacun, il est possible de mutualiser le traitement entre eux.

[1] https ://www.oasis-open.org/ (visité le 01/10/2018)
[2] https ://cloudify.co/ (visité le 01/10/2018)
[3] https ://ubicity.com/ (visité le 01/10/2018)

Cela peut permettre de réaliser un gain considérable en particulier pour les applications de l'IdO qui sont gourmandes en ressources (par exemple, traitement vidéo).

6.5 Coopération entre différents gestionnaires de services

Différentes études ont traité le placement sur les infrastructures géo-distribuées. Plusieurs d'entre elles ont ciblé des services spécifiques. Par exemple, Luizelli et al. ont traité le placement de chaînes de fonctions réseau virtuelles [Luizelli et al. 2015], Cardellini et al. ont traité le placement d'un service de traitement de flux pour la fouille de données [Cardellini et al. 2017] et Haghighi et al. ont traité le placement d'un réseau de diffusion de contenu [Haghighi et al. 2018]. En considérant la spécificité de ces services, il est possible d'améliorer les performances du placement. À titre d'exemple, dans notre cas, nous avons exploité le fait que les utilisateurs peuvent être intéressés par l'état de la même ressource afin de mutualiser les fonctions et les flux du service de supervision. Ainsi, nous avons réduit l'empreinte de calcul et réseau sur l'infrastructure Edge. Cependant, différents services (par exemple, service de supervision, service de réseau de diffusion de contenu, chaînes de fonctions réseau) peuvent être hébergés sur la même infrastructure. Dans ce cas, leurs calculateurs de placement doivent coordonner leurs tâches pour partager les ressources de l'infrastructure. Cela nécessite d'établir une stratégie de coopération entre eux.

Une solution possible est de modéliser la coopération entre eux par un problème de partitionnement de graphe où l'infrastructure est modélisée par un graphe à subdiviser [Afanasyev et al. 2018] entre les différents calculateurs de placement.

Références

Bibliographie

5G PPP (2015). *The 5G infrastructure public private partnership : The next generation of communication networks and services*.

Abderrahim, M., M. Ouzzif, K. Guillouard, J. Francois et A. Lebre (août 2017). « A Holistic Monitoring Service for Fog/Edge Infrastructures : A Foresight Study ». Dans : *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, p. 337-344. doi : 10.1109/FiCloud.2017.30.

Abdulrahim, Mohammad Abdulkader (1998). « Parallel Algorithms for Labeled Graph Matching ». AAI0599838. Thèse de doct. Golden, CO, USA.

Afanasyev, Andrey, Z Zhiming Zhao et A Arie Taal (2018). « Virtual infrastructure partitioning and provisioning under nearly real-time constraints ». Dans :

Ahmed, A. et E. Ahmed (2016). « A survey on mobile edge computing ». Dans : *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, p. 1-8. doi : 10.1109/ISCO.2016.7727082.

Andersen, Michael P et David E. Culler (2016). « BTrDB : Optimizing Storage System Design for Timeseries Processing ». Dans : *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA : USENIX Association, p. 39-52. isbn : 978-1-931971-28-7.

Antunes, Mário, Diogo Gomes et Rui L. Aguiar (2018). « Towards IoT data classification through semantic features ». Dans : *Future Generation Computer Systems* 86, p. 792 -798. issn : 0167-739X. doi : https://doi.org/10.1016/j.future.2017.11.045.

Arasu, Arvind, Shivnath Babu et Jennifer Widom (juin 2006). « The CQL continuous query language : semantic foundations and query execution ». Dans : *The VLDB Journal* 15.2, p. 121-142. issn : 0949-877X. doi : 10.1007/s00778-004-0147-z.

Asano, Takao et Tomio Hirata (1982). « Edge-deletion and Edge-contraction Problems ». Dans : *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. STOC '82. New York, NY, USA : ACM, p. 245-254. isbn : 0-89791-070-2. doi : 10.1145/800070.802198.

Assunção, Marcos Dias de, Alexandre da Silva Veith et Rajkumar Buyya (2018). « Distributed data stream processing and edge computing : A survey on resource elasticity and future directions ». Dans : *Journal of Network and Computer Applications* 103, p. 1 -17. issn : 1084-8045. doi : https://doi.org/10.1016/j.jnca.2017.12.001.

Asta, A (2016). *Observability at Twitter : technical overview, part i, 2016*. https://blog.twitter.com/engineering/en_us/a/2016/observability-at-twitter-technical-overview-part-i.html. Visité le : 2018-04-01.

Benazzouz, Y., C. Munilla, O. Günalp, M. Gallissot et L. Gürgen (mar. 2014). « Sharing user IoT devices in the cloud ». Dans : *2014 IEEE World Forum on Internet of Things (WF-IoT)*, p. 373-374. doi : 10.1109/WF-IoT.2014.6803193.

Bengoetxea, E. (2002). « Inexact Graph Matching Using Estimation of Distribution Algorithms ». Thèse de doct. Paris, France : Ecole Nationale Supérieure des Télécommunications. Chap. 10, p. 266-290.

Bernhardt, Thomas et Alexandre Vasseur (2007). « Esper : Event stream processing and correlation ». Dans : *ONJava, in http ://www. onjava. com/lpt/a/6955, O'Reilly*.

Binz, Tobias, Uwe Breitenbücher, Florian Haupt, Oliver Kopp, Frank Leymann, Alexander Nowak et Sebastian Wagner (2013). « OpenTOSCA – A Runtime for TOSCA-Based Cloud Applications ». Dans : *Service-Oriented Computing*. Sous la dir. de Samik Basu, Cesare Pautasso, Liang Zhang et Xiang Fu. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 692-695. isbn : 978-3-642-45005-1.

Boletsis, Costas et Dimitra Chasanidou (2018). « Audio Augmented Reality in Public Transport for Exploring Tourist Sites ». Dans : *Proceedings of the 10th Nordic Conference on Human-Computer Interaction*. NordiCHI '18. New York, NY, USA : ACM, p. 721-725. isbn : 978-1-4503-6437-9. doi : 10.1145/3240167.3240243.

Bonomi, Flavio, Rodolfo Milito, Jiang Zhu et Sateesh Addepalli (2012). « Fog computing and its role in the internet of things ». Dans : *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC '12. New York, NY, USA : ACM, p. 13-16. isbn : 978-1-4503-1519-7. doi : 10.1145/2342509.2342513.

Bousselmi, Ayoub, Sofiane Imadali et Marion Duprez (2018). *Plug'in : A 5G Experimentation Platform for Collaborative and Reproducible Research*. http://10.226.226.55/lucy/wp/files/1570456751.pdf. (Visité le : 2018-10-01).

Boykin, Oscar, Sam Ritchie, Ian O'Connell et Jimmy Lin (août 2014). « Summingbird : A Framework for Integrating Batch and Online MapReduce Computations ». Dans : *Proc. VLDB Endow.* 7.13, p. 1441-1451. issn : 2150-8097. doi : 10.14778/2733004.2733016.

Campolo, C., A. Molinaro, G. Araniti et A. O. Berthet (mar. 2017). « Better Platooning Control Toward Autonomous Driving : An LTE Device-to-Device Communications Strategy That Meets Ultralow Latency Requirements ». Dans : *IEEE*

Vehicular Technology Magazine 12.1, p. 30-38. ISSN : 1556-6072. DOI : 10.1109/MVT.2016.2632418.

CARDELLINI, Valeria, Vincenzo GRASSI, Francesco LO PRESTI et Matteo NARDELLI (mai 2017). « Optimal Operator Replication and Placement for Distributed Stream Processing Systems ». Dans : *SIGMETRICS Perform. Eval. Rev.* 44.4, p. 11-22. ISSN : 0163-5999. DOI : 10.1145/3092819.3092823.

CHAKRAVARTHY, Sharma et Qingchun JIANG (2009). *Stream Data Processing : A Quality of Service Perspective Modeling, Scheduling, Load Shedding, and Complex Event Processing*. 1st. Springer Publishing Company, Incorporated. ISBN : 0387710027, 9780387710020.

CHECKO, A., H. L. CHRISTIANSEN, Y. YAN, L. SCOLARI, G. KARDARAS, M. S. BERGER et L. DITTMANN (2015). « Cloud RAN for Mobile Networks—A Technology Overview ». Dans : *IEEE Communications Surveys Tutorials* 17.1, p. 405-426. ISSN : 1553-877X. DOI : 10.1109/COMST.2014.2355255.

CHOWDHURY, S. R., M. F. BARI, R. AHMED et R. BOUTABA (mai 2014). « PayLess : A low cost network monitoring framework for Software Defined Networks ». Dans : *2014 IEEE Network Operations and Management Symposium (NOMS)*, p. 1-9. DOI : 10.1109/NOMS.2014.6838227.

CHOY, Sharon, Bernard WONG, Gwendal SIMON et Catherine ROSENBERG (oct. 2014). « A Hybrid Edge-cloud Architecture for Reducing On-demand Gaming Latency ». Dans : *Multimedia Syst.* 20.5, p. 503-519. ISSN : 0942-4962. DOI : 10.1007/s00530-014-0367-z.

COMPUTING, Autonomic (2003). « An architectural blueprint for autonomic computing ». Dans : *IBM Publication*.

CONFAIS, B., A. LEBRE et B. PARREIN (déc. 2016). « Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures ». Dans : *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, p. 294-301. DOI : 10.1109/CloudCom.2016.0055.

DAVID, Pierre-Charles et Thomas LEDOUX (2005). « WildCAT : A Generic Framework for Context-aware Applications ». Dans : *Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing*. MPAC '05. New York, NY, USA : ACM, p. 1-7. ISBN : 1-59593-268-2. DOI : 10.1145/1101480.1101483.

DERI, Luca et Stefano SUIN (2000). « Effective traffic measurement using ntop ». Dans : *IEEE Communications Magazine* 38.5, p. 138-143.

DINH, Hoang T, Chonho LEE, Dusit NIYATO et Ping WANG (2013). « A survey of mobile cloud computing : architecture, applications, and approaches ». Dans : *Wireless communications and mobile computing* 13.18, p. 1587-1611.

Erbe, Heinz-H. (2004). « On Human Robot Collaboration - A Survey on Cost Aspects ». Dans : *IFAC Proceedings Volumes* 37.5. 7th IFAC Symposium on Cost-Oriented Automation (COA 2004), Gatineau, Québec, Canada, 6-9 June 2004, p. 167 -172. issn : 1474-6670. doi : https://doi.org/10.1016/S1474-6670(17)32361-3.

ESPN (2017). *Pay-per-view issues delay start of Floyd Mayweather-Conor McGregor fight*. http://www.espn.com/boxing/story/_/id/20469815/floyd-mayweather-conor-mcgregor-delay-ppv-problems. (Visité le : 2018-06-28).

Fang, X., S. Misra, G. Xue et D. Yang (avr. 2012). « Smart Grid — The New and Improved Power Grid : A Survey ». Dans : *IEEE Communications Surveys Tutorials* 14.4, p. 944-980. issn : 1553-877X. doi : 10.1109/SURV.2011.101911.00087.

Fletcher, Rick et Prakash Banthia (2000). *Distributed remote monitoring (dRMON) for networks*. US Patent 6,108,782.

Freire, Marília, Uirá Kulesza, Eduardo Aranha, Gustavo Nery, Daniel Costa, Andreas Jedlitschka, Edmilson Campos, Silvia T. Acuña et Marta N. Gómez (2014). « Assessing and Evolving a Domain Specific Language for Formalizing Software Engineering Experiments : An Empirical Study ». Dans : *International Journal of Software Engineering and Knowledge Engineering* 24.10, p. 1509-1531. doi : 10.1142/S0218194014400178. eprint : https://doi.org/10.1142/S0218194014400178.

Ghijsen, Mattijs, Jeroen van der Ham, Paola Grosso, Cosmin Dumitru, Hao Zhu, Zhiming Zhao et Cees de Laat (2013). « A semantic-web approach for modeling computing infrastructures ». Dans : *Computers & Electrical Engineering* 39.8, p. 2553 -2565. issn : 0045-7906. doi : https://doi.org/10.1016/j.compeleceng.2013.08.011.

Greer, Christopher, David A Wollman, Dean E Prochaska, Paul A Boynton, Jeffrey A Mazer, Cuong T Nguyen, Gerald J FitzPatrick, Thomas L Nelson, Galen H Koepke, Allen R Hefner Jr et al. (2014). *NIST framework and roadmap for smart grid interoperability standards, release 3.0*. Rapp. tech.

Gutierrez-Aguado, Juan, Jose M. Alcaraz Calero et Wladimiro Diaz Villanueva (juin 2016). « IaaSMon : Monitoring Architecture for Public Cloud Computing Data Centers ». Dans : *Journal of Grid Computing* 14.2, p. 283-297. doi : 10.1007/s10723-015-9357-4.

Ha, K., P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies et M. Satyanarayanan (mar. 2013). « The Impact of Mobile Multimedia Applications on Data Center Consolidation ». Dans : *2013 IEEE International Conference on Cloud Engineering (IC2E)*, p. 166-176. doi : 10.1109/IC2E.2013.17.

Haghighi, A. A., S. Shah Heydari et S. Shahbazpanahi (2018). « Dynamic QoS-Aware Resource Assignment in Cloud-Based Content-Delivery Networks ». Dans : *IEEE Access* 6, p. 2298-2309. issn : 2169-3536. doi : 10.1109/ACCESS.2017.2782776.

Halpern, Joel et Carlos Pignataro (2015). *Service function chaining (sfc) architecture*. Rapp. tech.

Ham, Jeroen van der, Freek Dijkstra, Roman Łapacz, Jason Zurawski et al. (2013). « Network markup language base schema version 1 ». Dans : Muncie, INOpen Grid Forum.

Hirzel, Martin, Robert Soulé, Scott Schneider, Buğra Gedik et Robert Grimm (mar. 2014). « A Catalog of Stream Processing Optimizations ». Dans : *ACM Comput. Surv.* 46.4, 46 :1-46 :34. issn : 0360-0300. doi : 10.1145/2528412.

Hoffman, Hunter G, Walter J Meyer III, Maribel Ramirez, Linda Roberts, Eric J Seibel, Barbara Atzori, Sam R Sharar et David R Patterson (2014). « Feasibility of articulated arm mounted Oculus Rift Virtual Reality goggles for adjunctive pain control during occupational therapy in pediatric burn patients ». Dans : *Cyberpsychology, Behavior, and Social Networking* 17.6, p. 397-401.

Jernigan, Stephanie, David Kiron et Sam Ransbotham (2016). « Data Sharing and Analytics are Driving Success With IoT ». Dans : *MIT Sloan Management Review* 58.1.

Jussien, Narendra, Guillaume Rochart et Xavier Lorca (2008). « Choco : an Open Source Java Constraint Programming Library ». Dans : *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*. Paris, France, France, p. 1-10.

Kahraman, Gökhan et Semih Bilgen (oct. 2015). « A framework for qualitative assessment of domain-specific languages ». Dans : *Software & Systems Modeling* 14.4, p. 1505-1526. issn : 1619-1374. doi : 10.1007/s10270-013-0387-8.

Kempe, D., A. Dobra et J. Gehrke (oct. 2003). « Gossip-based computation of aggregate information ». Dans : *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* P. 482-491. doi : 10.1109/SFCS.2003.1238221.

Åkerberg, J., M. Gidlund et M. Björkman (juil. 2011). « Future research challenges in wireless sensor and actuator networks targeting industrial automation ». Dans : *2011 9th IEEE International Conference on Industrial Informatics*, p. 410-415. doi : 10.1109/INDIN.2011.6034912.

Khodashenas, Pouria Sayyad, Cristina Ruiz, Muhammad Shuaib Siddiqui, August Betzler et Jordi Ferrer Riera (2017). « The role of Edge Computing in future 5G mobile networks : concept and challenges ». Dans : *Cloud and Fog Computing in 5G Mobile Networks : Emerging Advances and Applications* 70, p. 349.

Kumar, Vipin (avr. 1992). « Algorithms for Constraint-satisfaction Problems : A Survey ». Dans : *AI Mag.* 13.1, p. 32-44. issn : 0738-4602.

Kundu, Dinangkur et SM Ibrahim Lavlu (2009). *Cacti 0.8 network monitoring*. Packt Publishing Ltd.

Lebre, A., J. Pastor, A. Simonet et F. Desprez (avr. 2017). « Revising OpenStack to Operate Fog/Edge Computing Infrastructures ». Dans : *2017 IEEE International Conference on Cloud Engineering (IC2E)*, p. 138-148. doi : 10.1109/IC2E.2017.35.

Lecoutre, Christophe, Lakhdar Saïs, Sébastien Tabary et Vincent Vidal (déc. 2009). « Reasoning from Last Conflict(s) in Constraint Programming ». Dans : *Artif. Intell.* 173.18, p. 1592-1614. issn : 0004-3702. doi : 10.1016/j.artint.2009.09.002.

Lee, DongWoo, Jack J. Dongarra et R. S. Ramakrishna (2003). « visPerf : Monitoring Tool for Grid Computing ». Dans : *Computational Science — ICCS 2003*. Sous la dir. de Peter M. A. Sloot, David Abramson, Alexander V. Bogdanov, Yuriy E. Gorbachev, Jack J. Dongarra et Albert Y. Zomaya. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 233-243. isbn : 978-3-540-44863-1.

Legrand, I., H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, C. Dobre, A. Muraru, A. Costan, M. Dediu et C. Stratan (2009). « MonALISA : An agent based, dynamic service system to monitor, control and optimize distributed systems ». Dans : *Computer Physics Communications* 180.12. 40 YEARS OF CPC : A celebratory issue focused on quality software for high performance, grid and novel computing architectures, p. 2472 -2498. issn : 0010-4655. doi : https://doi.org/10.1016/j.cpc.2009.08.003.

Leighton, Tom (fév. 2009). « Improving Performance on the Internet ». Dans : *Commun. ACM* 52.2, p. 44-51. issn : 0001-0782. doi : 10.1145/1461928.1461944.

Luizelli, M. C., L. R. Bays, L. S. Buriol, M. P. Barcellos et L. P. Gaspary (mai 2015). « Piecing together the NFV provisioning puzzle : Efficient placement and chaining of virtual network functions ». Dans : *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, p. 98-106. doi : 10.1109/INM.2015.7140281.

Marescaux, Jaques et Francesco Rubino (2006). « Transcontinental Robot-Assisted Remote Telesurgery, Feasibility and Potential Applications ». Dans : sous la dir. de Kanagasingam Yogesan, Sajeesh Kumar, Leonard Goldschmidt et Jorge Cuadros, p. 261-265. doi : 10.1007/3-540-33714-8_31.

Massie, Matthew L, Brent N Chun et David E Culler (2004). « The ganglia distributed monitoring system : design, implementation, and experience ». Dans : *Parallel Computing* 30.7, p. 817 -840. issn : 0167-8191. doi : https://doi.org/10.1016/j.parco.2004.04.001.

Mastroianni, Carlo, Domenico Talia et Oreste Verta (2008). « Designing an information system for Grids : Comparing hierarchical, decentralized P2P and super-peer

models ». Dans : *Parallel Computing* 34.10, p. 593 -611. ISSN : 0167-8191. DOI : https://doi.org/10.1016/j.parco.2008.07.001.

MIJUMBI, R., J. SERRAT, J. GORRICHO, N. BOUTEN, F. De TURCK et R. BOUTABA (2016). « Network Function Virtualization : State-of-the-Art and Research Challenges ». Dans : *IEEE Communications Surveys Tutorials* 18.1, p. 236-262. ISSN : 1553-877X. DOI : 10.1109/COMST.2015.2477041.

MINOLI, D., K. SOHRABY et B. OCCHIOGROSSO (fév. 2017). « IoT Considerations, Requirements, and Architectures for Smart Buildings—Energy Optimization and Next-Generation Building Management Systems ». Dans : *IEEE Internet of Things Journal* 4.1, p. 269-283. ISSN : 2327-4662. DOI : 10.1109/JIOT.2017.2647881.

MORIN, B., T. LEDOUX, M. B. HASSINE, F. CHAUVEL, O. BARAIS et J. JEZEQUEL (oct. 2009). « Unifying Runtime Adaptation and Design Evolution ». Dans : *2009 Ninth IEEE International Conference on Computer and Information Technology*. T. 1, p. 104-109. DOI : 10.1109/CIT.2009.94.

NAGIOS (2014). *Nagios – Agent Comparison*. https://assets.nagios.com/downloads/ncpa/docs/NCPA-Agent-Comparison.pdf. (Visité le : 2018-06-28).

NEWHALL, Tia, Janis LIBEKS, Ross GREENWOOD et Jeff KNERR (2010). « PeerMon : A Peer-to-peer Network Monitoring System ». Dans : *Proceedings of the 24th International Conference on Large Installation System Administration*. LISA'10. Berkeley, CA, USA : USENIX Association, p. 1-12.

OLAVERRI-MONREAL, C., P. GOMES, R. FERNANDES, F. VIEIRA et M. FERREIRA (juin 2010). « The See-Through System : A VANET-enabled assistant for overtaking maneuvers ». Dans : *2010 IEEE Intelligent Vehicles Symposium*, p. 123-128. DOI : 10.1109/IVS.2010.5548020.

ONF (mar. 2015). *OpenFlow Switch Specification*.

PANG, Zhibo (2013). « Technologies and Architectures of the Internet-of-Things (IoT) for Health and Well-being ». QC 20130523. Thèse de doct. KTH, Electronic Systems, p. xiv, 75. ISBN : 978-91-7501-736-5.

PARVEZ, I., A. RAHMATI, I. GUVENC, A. I. SARWAT et H. DAI (2018). « A Survey on Low Latency Towards 5G : RAN, Core Network and Caching Solutions ». Dans : *IEEE Communications Surveys Tutorials*, p. 1-1. ISSN : 1553-877X. DOI : 10.1109/COMST.2018.2841349.

PATHAN, A. et R. BUYYA (fév. 2007). « A taxonomy and survey of content delivery networks, » dans :

PERSSON, Per et Ola ANGELSMARK (2015). « Calvin – Merging Cloud and IoT ». Dans : *Procedia Computer Science* 52. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference

on Sustainable Energy Information Technology (SEIT-2015), p. 210 -217. ISSN : 1877-0509. DOI : https://doi.org/10.1016/j.procs.2015.05.059.

PING, Su (2003). « Delay measurement time synchronization for wireless sensor networks ». Dans : *Intel Research Berkeley Lab* 6, p. 1-12.

PRATHIBA, S. et S. SOWVARNICA (fév. 2017). « Survey of failures and fault tolerance in cloud ». Dans : *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, p. 169-172. DOI : 10.1109/ICCCT2.2017.7972271.

PRIES, R., H. MORPER, N. GALAMBOSI et M. JARSCHEL (sept. 2016). « Network as a Service - A Demo on 5G Network Slicing ». Dans : *2016 28th International Teletraffic Congress (ITC 28)*. T. 01, p. 209-211. DOI : 10.1109/ITC-28.2016.136.

RENATER (2007). *IPv4 MULTICAST Service*. https://pasillo.renater.fr/test/get_qosmetrics_resultsMULTICASTv4.php. (Visité le : 2018-03-08).

ROGERS, Ethan A (2014). « The Energy Savings Potential of Smart Manufacturing ». Dans : American Council for an Energy-Efficient Economy.

ROLFFS, Paula, David OCKWELL et Rob BYRNE (2015). « Beyond technology and finance : pay-as-you-go sustainable energy access and theories of social change ». Dans : *Environment and Planning A : Economy and Space* 47.12, p. 2609-2627. DOI : 10.1177/0308518X15615368. eprint : https://doi.org/10.1177/0308518X15615368.

ROMANO, L., D. De MARI, Z. JERZAK et C. FETZER (juin 2011). « A Novel Approach to QoS Monitoring in the Cloud ». Dans : *2011 First International Conference on Data Compression, Communications and Processing*, p. 45-51. DOI : 10.1109/CCP.2011.49.

ROWSTRON, Antony et Peter DRUSCHEL (2001). « Pastry : Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems ». Dans : *Middleware 2001*. Sous la dir. de Rachid GUERRAOUI. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 329-350. ISBN : 978-3-540-45518-9.

RUMBLE, Stephen M., Diego ONGARO, Ryan STUTSMAN, Mendel ROSENBLUM et John K. OUSTERHOUT (2011). « It's Time for Low Latency ». Dans : *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*. HotOS'13. Berkeley, CA, USA : USENIX Association, p. 11-11.

SALVADOR, Paulo et Rui VALADAS (2005). « A network monitoring system with a Peer-to-Peer architecture ». Dans : *Proceedings of the thrid international workshop on internet performance, simulation, monitoring and measurements (IPS-MoMe 2005)*. Citeseer, p. 14-15.

SCHAFER, Valérie (2015). « Part of a whole : RENATER, a twenty-year-old network within the Internet ». Dans : *Information & Culture* 50.2, p. 217-235.

Seok, Seung&#45 ;Joon, Wang&#45 ;Cheol Song et Deokjai Choi (oct. 2010). « Implementation of Pastry&#45 ;Based P2P System to Share Sensor Data ». Dans : *Int. J. Sen. Netw.* 8.3/4, p. 125-135. issn : 1748-1279. doi : 10.1504/IJSNET.2010.036188.

Shi, W., J. Cao, Q. Zhang, Y. Li et L. Xu (oct. 2016). « Edge Computing : Vision and Challenges ». Dans : *IEEE Internet of Things Journal* 3.5, p. 637-646. issn : 2327-4662. doi : 10.1109/JIOT.2016.2579198.

Smit, Michael, Bradley Simmons et Marin Litoiu (oct. 2013). « Distributed, Application-level Monitoring for Heterogeneous Clouds Using Stream Processing ». Dans : *Future Gener. Comput. Syst.* 29.8, p. 2103-2114. issn : 0167-739X. doi : 10.1016/j.future.2013.01.009.

Spinnewyn, Bart, Ruben Mennes, Juan Felipe Botero et Steven Latré (2017). « Resilient application placement for geo-distributed cloud networks ». Dans : *Journal of Network and Computer Applications* 85. Intelligent Systems for Heterogeneous Networks, p. 14 -31. issn : 1084-8045. doi : https://doi.org/10.1016/j.jnca.2016.12.015.

Stallings, William (1998). *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc.

Stone, Robert J. (2001). « Haptic feedback : a brief history from telepresence to virtual reality ». Dans : *Haptic Human-Computer Interaction*. Sous la dir. de Stephen Brewster et Roderick Murray-Smith. Berlin, Heidelberg : Springer Berlin Heidelberg, p. 1-16. isbn : 978-3-540-44589-0.

Subramaniyan, Rajagopal, Pirabhu Raman, Alan D. George et Matthew Radlinski (2006). « GEMS : Gossip-Enabled Monitoring Service for Scalable Heterogeneous Distributed Systems ». Dans : *Cluster Computing* 9.1, p. 101-120. issn : 1573-7543. doi : 10.1007/s10586-006-4900-5.

Suneja, Sahil, Canturk Isci, Vasanth Bala, Eyal de Lara et Todd Mummert (juin 2014). « Non-intrusive, Out-of-band and Out-of-the-box Systems Monitoring in the Cloud ». Dans : t. 42. 1. New York, NY, USA : ACM, p. 249-261. doi : 10.1145/2637364.2592009.

Synergy, The Research Group (2018). *2017 Review Shows $180 billion Cloud Market Growing at 24% Annually*. https://www.srgresearch.com/articles/2017-review-shows-180-billion-cloud-market-growing-24-annually. (Visité le : 2018-06-28).

Tastevin, N., M. Obadia et M. Bouet (mai 2017). « A graph approach to placement of Service Functions Chains ». Dans : *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, p. 134-141. doi : 10.23919/INM.2017.7987273.

Toshniwal, Ankit, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Don-

HAM, Nikunj BHAGAT, Sailesh MITTAL et Dmitriy RYABOY (2014). « Storm@Twitter ». Dans : *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. SIGMOD '14. Snowbird, Utah, USA : ACM, p. 147-156. ISBN : 978-1-4503-2376-5. DOI : 10.1145/2588555.2595641.

VERFAILLIE, Gérard et Narendra JUSSIEN (juil. 2005). « Constraint Solving in Uncertain and Dynamic Environments : A Survey ». Dans : *Constraints* 10.3, p. 253-281. ISSN : 1572-9354. DOI : 10.1007/s10601-005-2239-9.

WANG, S., X. ZHANG, Y. ZHANG, L. WANG, J. YANG et W. WANG (2017). « A Survey on Mobile Edge Networks : Convergence of Computing, Caching and Communications ». Dans : *IEEE Access* 5, p. 6757-6779. ISSN : 2169-3536. DOI : 10.1109/ACCESS.2017.2685434.

WARD, J. S. et A. BARKER (déc. 2013). « Varanus : In Situ Monitoring for Large Scale Cloud Systems ». Dans : *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. T. 2, p. 341-344. DOI : 10.1109/CloudCom.2013.164.

WARD, Jonathan Stuart et Adam BARKER (déc. 2014). « Observing the clouds : a survey and taxonomy of cloud monitoring ». Dans : *Journal of Cloud Computing* 3.1, p. 24. ISSN : 2192-113X. DOI : 10.1186/s13677-014-0024-2.

XU, B., L. D. XU, H. CAI, C. XIE, J. HU et F. BU (mai 2014a). « Ubiquitous Data Accessing Method in IoT-Based Information System for Emergency Medical Services ». Dans : *IEEE Transactions on Industrial Informatics* 10.2, p. 1578-1586. ISSN : 1551-3203. DOI : 10.1109/TII.2014.2306382.

XU, Mengyuan, Zhihan LIU et Jinglin LI (2014b). « Tree-Structured Network Based Hierarchical Complex Event Processing in Wireless Sensor Networks ». Dans : *2014 Asia-Pacific Services Computing Conference (APSCC)*. IEEE, p. 185-190.

YU, Yuan, Pradeep Kumar GUNDA et Michael ISARD (2009). « Distributed Aggregation for Data-parallel Computing : Interfaces and Implementations ». Dans : *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*. SOSP '09. New York, NY, USA : ACM, p. 247-260. ISBN : 978-1-60558-752-3. DOI : 10.1145/1629575.1629600.

ZHANG, Ben, Nitesh MOR, John KOLB, Douglas S. CHAN, Ken LUTZ, Eric ALLMAN, John WAWRZYNEK, Edward LEE et John KUBIATOWICZ (2015). « The Cloud is Not Enough : Saving IoT from the Cloud ». Dans : *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*. Santa Clara, CA : USENIX Association.

ZHANG, Gang, Xiaoyan KUANG, Jing CHEN et Yu ZHANG (juil. 2009). « Design and implementation of a leader election algorithm in hierarchy mobile ad hoc network ». Dans : *2009 4th International Conference on Computer Science Education*, p. 263-268. DOI : 10.1109/ICCSE.2009.5228448.

Zhang, Guohui et Yinhai Wang (2013). « Optimizing coordinated ramp metering : a preemptive hierarchical control approach ». Dans : *Computer-Aided Civil and Infrastructure Engineering* 28.1, p. 22-37. doi : 10.1111/j.1467-8667.2012.00764.x. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8667.2012.00764.x.

Zhang, J., X. Hu, Z. Ning, E. C. . Ngai, L. Zhou, J. Wei, J. Cheng et B. Hu (août 2018). « Energy-Latency Tradeoff for Energy-Aware Offloading in Mobile Edge Computing Networks ». Dans : *IEEE Internet of Things Journal* 5.4, p. 2633-2645. issn : 2327-4662. doi : 10.1109/JIOT.2017.2786343.

Table des figures

Liste des tableaux

Acronymes

5G Réseaux de la 5ème génération. 11, 105

AN Acess Node ou Nœud d'accès. 25

API Interface de programmation (Application Programming Interface). 33, 36, 39–42, 45, 48, 50, 66

CMC Cloud Monitoring Checks. 46

CPU Central Processing Unit. 25, 41, 42, 53, 54, 64

ED Edge Device ou Équipement à la périphérie du réseau. 25–27

gmetad Ganglia Meta Daemon. 46, 47

gmond Ganglia Monitoring Daemon. 46

I/O Input/Output ou Entrées-sorties. 26, 68, 70

IaaS Infrastructure as a Service. 19

ICMP Internet Control Message Protocol. 32

IdO Internet des objets. 20–22, 76, 110, 111

IMM Infrastructure Monitoring Manager. 46

INDL Infrastructure and Network Description Language. 69–71, 127

IT Information Technology ou technologie d'information. 34, 66, 69

LAN Local Area Network. 41

MIB Management Information Base ou Base d'information pour la gestion. 40

NCPA Nagios Cross Platform Agent. 45

NFM Near Field Monitoring. 39, 57, 127

NFV Network Function Virtualization ou virtualisation des fonctions réseau. 26

NML Network Markup Language. 69, 70, 127

NMS Network Management Station ou Station de gestion réseau. 40

OASIS Organisation pour l'évolution des normes d'information structurée. 110

PaaS Platform as a Service. 18, 69

PoP Point of Presence ou Point de présence. 25, 26, 73

QoE Quality of Experience ou Qualité d'expérience. 25

QoS Quality of Service ou Qualité de service. 67, 82–84, 89

RAM Random-Access Memory. 41, 54, 64, 68, 72, 73, 92

REST Representational State Transfer. 32, 50

RTT Round Trip Time. 23, 24, 129

SaaS Software as a Service. 18

SDN Software Defined Network ou Réseau programmable. 40

SLA Service Level Agreement ou contrat sur le niveau de service. 18, 99

SMS Short Message Service. 33, 66

SNMP Simple Network Management Protocol. 40, 41

TCP Transmission Control Protocol. 23, 36

TTL Time To Live. 42, 43

UDP User Datagram Protocol. 36, 45

VoD Video on Demand ou Vidéos à la demande. 72

Wi-Fi Wireless local area networking. 26, 68

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Conception d'un système de supervision programmable et reconfigurable pour une infrastructure informatique et réseau répartie

Mots clés : Edge, Supervision, Canevas logiciel, Placement, Mutualisation, Langage

Résumé : Le Cloud offre le calcul, stockage et réseau en tant que services. Pour réduire le coût de cette offre, les opérateurs ont tendance à s'appuyer sur des infrastructures centralisées et gigantesques. Cependant, cette configuration entrave la satisfaction des exigences de latence et de bande passante des applications de nouvelle génération. L'Edge cherche à relever ce défi en s'appuyant sur des ressources massivement distribuées. Afin de satisfaire les attentes des opérateurs et des utilisateurs du Edge, des services de gestion ayant des capacités similaires à celles qui ont permis le succès du Cloud doivent être conçus. Dans cette thèse, nous nous concentrons sur le service de supervision. Nous proposons un canevas logiciel pour la mise en place d'un service holistique. Ce canevas permet de déterminer une architecture de déploiement pair-à-pair pour les fonctions d'observation, de traitement et d'exposition des mesures. Il vérifie que cette architecture satisfait les exigences fonctionnelles et de qualité de service des utilisateurs. Ces derniers peuvent être exprimés à l'aide d'un langage de description offert par le canevas. Le canevas offre également un langage de description pour unifier la description de l'infrastructure Edge. L'architecture de déploiement est déterminée avec l'objectif de minimiser l'empreinte de calcul et réseau du service de supervision. Pour cela, les fonctions de supervision sont mutualisées entre les différents utilisateurs. Les tests que nous avons faits ont montré la capacité de notre proposition à réduire l'empreinte de supervision avec un gain qui atteint -28% pour le calcul et -24% pour le réseau.

Title : Toward a programmable and reconfigurable monitoring system for an edge infrastructure

Keywords : Edge, Monitoring, Framework, Placement, Mutualization, Language

Abstract : Cloud offers compute, storage and network as services. To reduce the offer cost, the operators tend to rely on centralized and massive infrastructures. However, such a configuration hinders the satisfaction of the latency and bandwidth requirements of new generation applications. The Edge aims to rise this challenge by relying on massively distributed resources. To satisfy the operators and the users of Edge, management services similar to the ones that made the success of Cloud should be designed. In this thesis, we focus on the monitoring service. We design a framework to establish a holistic monitoring service. This framework determines a peer-to-peer deployment architecture for the observation, processing, and exposition of measurements. It verifies that this architecture satisfies the functional and quality of service constraints of the users. For this purpose, it relies on a description of users requirements and a description of the Edge infrastructure. The expression of these two elements can be unified with two languages offered by the Framework. The deployment architecture is determined with the aim of minimizing the compute and network footprint of the monitoring service. For this purpose, the functions are mutualized as much as possible among the different users. The tests we did showed the relevance of our proposal for reducing monitoring footprint with a gain of -28% for the compute and -24% for the network.