Optimal Design and Resource Management of Fog Infrastructures for Vehicular Networks

Ioanna Stypsanelli

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *07/12/2020* **par :**

Ioanna Vasiliki STYPSANELLI

Optimal Design and Resource Management of Fog Infrastructures for Vehicular Networks

---

JURY

Congduc Pham	Professeur des universités	Rapporteur
Toufik Ahmed	Professeur des universités	Rapporteur
Thierry Monteil	Professeur des universités	Président du jury
Anne-Cécile Orgerie	Chargé de Recherche	Examinateur
Olivier Brun	Directeur de Recherche	Directeur de thèse
Balakrishna Prabhu	Chargé de Recherche	Directeur de thèse
Samir Medjiah	Maître de conférences	Membre Invité
Olivier Guérard	Ingénieur	Membre Invité

École doctorale et spécialité :

EDSYS : Informatique 4200018

Unité de Recherche :

LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Directeur(s) de Thèse :

Olivier Brun et *Balakrishna Prabhu*

Rapporteurs :

Congduc Pham et *Toufik Ahmed*

Abstract

The advent of connected cars will drive the need for infrastructures supporting latency critical applications whose requirements can no longer be supported by the distant Cloud. The Fog, on the other hand, is a new paradigm bringing computation, storage and networking closer to the user. The Fog can be seen as a more geographically distributed Cloud. However building such an infrastructure requires thoughtful design decisions to minimize the installation and operating costs. There are other costs associated with operating a Fog network that can be optimized by considering mobility patterns. There are also challenges around allocating tasks in Fog networks.

In this thesis, we provide solutions to minimize the Fog infrastructure costs including compute capacity, bandwidth and energy by taking advantage of traffic statistics and mobility patterns. In a first study, we explore how to minimize the cost of an upcoming Fog infrastructure at its design phase. The solution we propose uses optimization and queuing theory and returns the optimal set of Fog nodes to open conjointly with the corresponding routing strategy for base stations. In a second study, we provide a model to minimize overheads introduced by service migrations in the Fog based on mobility patterns. In the last study, we examine distributed task allocation strategies for base stations which neither cooperate nor require knowledge of the physical infrastructure. We propose a simulation model to compare algorithms reacting to the response times they observe from Fog nodes and provide benchmarks.

Abstract

L'avènement des véhicules connectés va entraîner la nécessité de construire des infrastructures capables de répondre aux besoins d'applications à latence faible dont les contraintes ne pourront plus être satisfaites par le Cloud. Le Fog est un nouveau paradigme qui rapproche les calculs, le stockage et le réseau de l'utilisateur. Le Fog peut être perçu comme un Cloud plus réparti géographiquement. Cependant, la construction d'une telle infrastructure nécessite une décision réfléchie afin de minimiser les coûts d'installation et d'opération. D'autres coûts associés au fonctionnement d'un réseau Fog peuvent être optimisés en fonction des profils de mobilité des usagers. Il y a également des défis autour de l'allocation des tâches dans les réseaux Fog.

Dans cette thèse, nous proposons des solutions pour minimiser les coûts d'une infrastructure Fog, y compris ceux de capacité de calcul, la bande passante et les coûts énergétiques, en tirant parti de statistiques de trafic et des profils de mobilité. Dans une première étude, nous explorons comment minimiser le coût d'une infrastructure Fog lors de sa conception. La solution que nous proposons utilise l'optimisation et la théorie des files d'attentes et permet d'obtenir un ensemble optimal de nœuds Fog à installer ainsi que la stratégie de routage des stations de base cellulaires adaptée. Dans une deuxième étude, nous proposons un modèle pour minimiser les surcoûts résultant des migrations de service à l'aide des profils de mobilité. Dans la dernière étude, nous examinons des stratégies réparties d'allocation de tâches pour les stations de base sans qu'elles n'aient besoin de coopérer ni qu'elles n'aient aucune information sur l'infrastructure. Nous proposons un modèle de simulation pour comparer la réaction de divers algorithmes en fonction des temps de réponse des nœuds Fog qu'ils observent.

Remerciements

Tout d'abord, je tiens à remercier mes directeurs de thèse Olivier Brun, Balakrishna Prabhu et Samir Medjiah qui m'ont donné l'opportunité de faire cette thèse. Je les remercie pour tout ce qu'ils m'ont appris.

Je remercie également Olivier Guérard, de la part de Continental, pour son accueil et l'enthousiasme de partager ses connaissances.

Je souhaite exprimer ma reconnaisance envers mes rapporteurs de thèse Congduc Pham et Toufik Ahmed pour avoir accepté d'évaluer cette thèse. Leurs remarques m'ont permis de réflechir à mes travaux d'une autre perspective. Merci à Anne-Cécile Orgerie et Thierry Monteil d'avoir également fait partie de mon jury de thèse et pour leur bienveillance.

Merci à Angelina Vidali, Mohamed Siala et Jose Aguilar pour les échanges chaleureux, leur enthousiasme et leur dynamisme qui m'ont inspirée.

Ce chapitre de ma vie a été marqué par l'ensemble des membres du LAAS que j'ai rencontré. C'était d'abord un grand privilège de faire partie de ce laboratoire très dynamique. Je remercie tous ceux que j'ai côtoyés, Je garde toujours un très bon souvenir de l'ensemble de l'équipe SARA, à commencer par mes collègues de bureau avec lesquels j'ai passé d'excellents moments: Nga, Benoit, Marine, Josue. Je remercie tous les doctorants de l'équipe d'être une compagnie sympathique lors de pauses repas et café. Je pense également à mes amis du LAAS Ilias, Ruth, Margot et Santiago pour les beaux moments et leur soutien, ainsi qu'aux amis rencontrés dans le cadre des TPs, Thibault pour les cafés matinaux au Busca et Guillaume.

Je remercie également mes amis Eleni (les trois!), Rina Teni, Xaroula, Maria et Grigoris (en pensant à Alex), Georgia, Giannis, Giorgos, Sofia, Matina, Anastasia et Maria, Denisa, Daniela, Irina, Antoine, Tomasz, Nikos, Constantin, Babis, Jim, Larry, les jumeaux et tous ceux du Kung-Fu qui m'ont aidée à me défouler.

Enfin, je remercie ma famille et grands-parents qui étaient très fiers de moi, ainsi que la famille que j'ai faite en France pour leur soutien ces années et leur aide très appréciée. Pour finir, je tiens à remercier mon *beloved* Simon de m'avoir soutenue, pour sa patience et pour son amour. Je te remercie infiniment.

Resumé en français

Contrairement aux services traditionnels déployés dans le Cloud, de nombreuses applications émergentes, issues notamment de l'IoT, ont besoin d'une latence faible et prédictible. C'est par exemple le cas des applications des véhicules connectés liées à la sécurité des automobilistes. Le Cloud offre une grande puissance de calcul, néanmoins les datacenters restent éloignés des utilisateurs.

Plusieurs paradigmes ont été étudiés cette dernière décennie afin de rapprocher le calcul et le stockage à la terminaison du réseau. Parmi ces initiatives on peut noter l'*Edge computing* qui permet de répartir davantage le calculs en limitant la nécessité d'utiliser le réseau jusqu'aux datacenters du Cloud. Le paradigme de l'Edge a été initialement utilisé pour la distribution de contenus (Content Delivery Networks, ou CDN). La position exacte de la terminaison du réseau dépend du point de vue du fournisseur et de l'utilisateur. Les utilisateurs ayant des opérateurs réseau différents, il n'y a pas de garantie que l'opérateur fournisse des services Edge au point de terminaison de leur réseau et dans ce cas il s'agit de rejoindre le point Edge le plus proche.

Un autre paradigme, le *Fog Computing*, a été proposé par Cisco en 2015 pour créer un continuum entre le Cloud et l'Edge et rend transparente la gestion des ressources en fonction de la qualité de service demandée. Plusieurs architectures Fog ont été proposées par rapport à la disposition des ressources. Certains imaginent les ressources computationnelles ou de stockage directement dans les stations de base. D'autres l'imaginent directement dans les appareils des utilisateurs. Une autre approche consiste à considérer que les ressources Fog sont dans les nœuds indépendants, similaires à des micro-datacenters (ou *Cloudlets* [Ceselli 2015]). Selon la définition officielle du NIST [NIS] ils sont appelés simplement des *nœuds Fogs* qui sont répartis sur le territoire à proximité immédiate des utilisateurs. C'est cette dernière approche que nous utilisons dans nos études.

Il est toutefois essentiel de garantir que les gains en termes de latence ne soient pas obtenus au prix d'une explosion des coûts de capacité, d'exploitation et d'énergie due à la duplication de ressources dans de multiples nœuds Fog. En d'autres termes, il est impératif de trouver un bon compromis entre le coût de

l'infrastructure de Fog Computing et le respect des contraintes de délai.

Problèmes étudiés

Nous nous intéresserons à trois problématiques. La première cherche à fournir une solution pour la conception d'infrastructures Fog qui est optimale en termes de coût et de qualité de service pour un réseau donné et une estimation de trafic prévu. Nous cherchons à fournir le meilleur compromis entre qualité de service et coût d'infrastructure. En effet fournir la meilleure qualité de service en multipliant les nœuds Fog serait une approche simple mais engendre un coût très important, à l'inverse centraliser les datacenters comme le fait le Cloud ne permet pas de garantir les latences requises à la qualité de service exigée par le Fog. Nous y répondons par une solution qui utilise les technique de l'optimisation linéaire mixte.

La deuxième problématique cherche à fournir des techniques pour minimiser le coût dû à la migration de services Fog et en particulier pour répondre aux événements liés à la mobilité des utilisateurs. On imagine que les utilisateurs du Fog, particulièrement ceux qui bénéficient des services du Fog à travers leur véhicule connecté, vont se déplacer de manière à ce que les latences promises vers un noeud Fog risquent de croître au-delà des contraintes de qualité de service et qu'il faudra migrer le traitement du service vers un noeud Fog plus proche de la nouvelle position des utilisateurs. Dans ce cas, nous considérons que cette migration de services va améliorer la qualité de service. On pourrait imaginer une solution dans laquelle les services suivent les utilisateurs et se déplacent donc à chaque fois que l'utilisateur s'éloigne. Néanmoins, les migrations fréquentes engendrent des coûts inutiles.

Nous étudions donc une solution pour minimiser le nombre des migrations tout en garantissant un niveau de qualité de service et ainsi donc satisfaire les exigences tout en minimisant le coût global de l'utilisation de l'infrastructure.

La dernière problématique se déroule également à la phase opérationnelle mais de façon continue. Il s'agit d'étudier la façon dont les stations de base allouent les tâches aux nœuds Fog et à quel nœud Fog lorsqu'ils sont connectés à plusieurs nœuds. Cette fois ci nous utilisons une approche orientée simulations avec un modèle non-coopératif ou les stations de base ne partagent pas d'information et n'ont pas besoin de communiquer. Nous examinons comment les tâches sont allouées par les stations de base qui ne font qu'observer les temps de réponse des services envoyées aux différents nœuds. Différents algorithmes sont comparés à l'aide de scénarios simulant des événement imprévus, comme des perturbations. Enfin nous nous penchons sur le problème d'offloading de tâches d'un nœud Fog

vers le Cloud.

Première Problématique

Le problème peut être formalisé de la façon suivante. On considère un ensemble $\mathcal{D}$ de sites potentiels où on peut installer un micro-datacentre, un ensemble $\mathcal{B}$ de noeuds source de trafic, et un ensemble $\mathcal{S}$ de classes de jobs. On connaît la demande $\lambda_i^{k,t}$ de chaque noeud source i et de chaque classe k à chaque instant $t = 1, 2, \ldots, \tau$. Il s'agit de prendre trois types de décision:

- Il faut décider quels sont les sites à ouvrir, sachant que le site j a un coût d'ouverture β_j. On définit pour cela la la variable binaire u_j qui indique si le site j est ouvert ($u_j = 1$).
- Il faut déterminer le nombre c_j de serveurs installés dans le site j. On note $g_j(c)$ le coût lié à l'installation de c serveurs dans le site j. Ce coût est en général une fonction concave de la capacité pour représenter les économies d'échelle en faveur des grands datacenters.
- Il faut décider du routage des trafics issus des différentes sources de trafic. On note $x_{i,j}^{k,t}$ la quantité de trafic de classe k de i vers j à l'instant t. Ces variables de routage doivent vérifier des contraintes linéaires classiques de conservation et des contraintes de positivité.

Chaque job de classe k doit être traité dans un délai maximum de T_k unités de temps. Plus précisément, si on note $S_j^{k,t}$ la variable aléatoire représentant le temps de traitement d'un job de classe k dans le site j à l'instant t, et $\ell_{i,j}^k$ le temps réseau (supposé fixe) pour envoyer la requête et recevoir la réponse, il s'agit de vérifier la contrainte probabiliste $\mathbb{P}\left(S_j^{k,t} + \ell_{i,j}^k \geq T_k\right) \leq \delta_k$. En d'autres termes, la probabilité que le temps de traitement de bout-en-bout d'une requête de classe k envoyée au site j soit plus grand que T_k doit être inférieure à un seuil fixé δ_k.

L'objectif est de minimiser le coût global de l'infrastructure

$$\sum_{j \in \mathcal{D}} \beta_j \, u_j \; + \; g_j(c_j),$$

tout en satisfaisant les contraintes probabilistes sur les délais des jobs et les contraintes de routage.

Résolution

Nous supposons que chaque type de jobs est traité sur des serveurs dédiés, et on note c_j^k le nombre de serveurs réservés pour les jobs de classe k dans le site j. On a alors $c_j = \sum_k c_j^k$. Nous supposons de plus que les serveurs dédiés à la classe k peuvent être modélisés par un ensemble de c_j^k files $M/M/1/\infty$, chaque nouvelle requête étant routée vers un de ces serveurs avec la probabilité $\frac{1}{c_j^k}$ (routage de Bernoulli). Sous ces hypothèses, on a $\mathbb{P}\left(S_j^{k,t} \geq z\right) = e^{-(\mu_k - y_j^{k,t}/c_j^k)\, z}$, où $1/\mu_k$ est le temps moyen de traitement d'un job de classe k sur un serveur, et où $y_j^{k,t}$ est le taux d'arrivée des requêtes de classe k au site j à l'époque $t = 1, \ldots, \tau$. En utilisant cette expression, on peut montrer que, étant donné un routage fixé du trafic, la valeur optimale de c_j^k est

$$c_j^k = \max_{t,i} \left\{ \frac{y_j^{k,t}}{\mu_k - d_{i,j}^k}\, a_{i,j}^{k,t} \right\}, \tag{1}$$

où $d_{i,j}^k = \log(\frac{1}{\delta_k}) / \left[T_k - \ell_{i,j}^k\right]$ est une constante, où la variable binaire $a_{i,j}^{k,t}$ vaut 1 si $x_{i,j}^{k,t} > 0$. Les constants $\ell_{i,j}^k$ expriment les latences des programmations fixes en entrée du programme.

Dans les deux cas, le problème peut alors être résolu en utilisant un solveur MILP.

L'équation (1) peut aisément être linéarisée, ce qui permet de formuler le problème comme un problème linéaire en nombres entiers dans le cas où $g_j(c) = \alpha_j\, c$. Dans le cas où la fonction $g()$ est une fonction non-linéaire concave, on peut utiliser une approximation linéaire par morceaux. Concrètement, le solveur MILP dans Gurobi que nous utilisons est capable de linéariser cette fonction à l'aide de piecewise linearity function et de façon tout à fait automatique grâce à un objectif dit "PWL" qui utilise des "Special Ordered Sets".

Conclusion de la premiere problèmatique

Les expérimentations réalisées avec Gurobi ont montré que des instances de problème réalistes peuvent être résolus en des temps de calcul raisonnables. Les comparaisons avec deux solutions heuristiques, consistant soit à router les trafics vers le micro-datacenter le plus proche, soit à utiliser la solution centralisée de coût minimum, suggèrent que des économies significatives peuvent être réalisées en utilisant la solution optimale du problème.

Deuxième Problématique

Un défi majeur qui doit être traité dans le Fog est de maintenir la qualité de service des utilisateurs mobiles. Dans cette deuxième problématique nous nous intéresserons aux *migrations de services* dans les infrastructures Fog. Une migration de service correspond au déplacement d'un service applicatif et éventuellement des données associées d'un nœud Fog à un autre. Elle a pour objectif de satisfaire les contraintes de latence et est en général nécessaire lorsqu'un utilisateur s'éloigne trop du nœud Fog qui lui a été affecté. En pratique, ces migrations ont un coût et plus elles sont fréquentes, plus le coût sera important pour l'opérateur réseau. Il faut noter par ailleurs que certains services sont *stateful* (le résultat d'une requête dépend du contexte et de l'historique), ce qui implique de restaurer l'état persistant lors de la réinstanciation du service, alors que d'autres sont *stateless* (chaque transaction est effectuée à partir de rien, comme si c'était la première fois). Il s'agit d'un sujet important qui n'est pas traité dans cette thèse.

Dans notre approche, le déplacement des données et du service sont considérés comme une seule migration. L'idée de notre étude est de trouver comment minimiser ces coûts en minimisant le nombre de migrations, tout en conservant la qualité de service requise. On peut imaginer une solution naïve dans laquelle le service est toujours migré vers le nœud Fog le plus proche de la station de base à laquelle l'utilisateur est connecté.

Cependant ce modèle peut engendrer des migrations inutiles et nous cherchons donc à trouver quels nœuds Fog les stations de base doivent privilégier afin d'éviter des migrations inutiles.

Dans un premier temps nous nous intéressons à l'optimisation pour toute une flotte des véhicules en partant des métriques qui ont été collectées par les stations de base sur les événements de handover du réseaux radio 4G/5G pour un réseau Fog donné.

Nous pré-sélectionnons les nœuds Fog qui satisfont les contraintes de qualité de service et de latence par chaque station de base. Notre objectif correspond donc à minimiser le coût tout en garantissant que toutes les stations de base sont connectées et peuvent allouer des tâches. Nous proposons une formulation linéaire en nombres entiers du problème, qui peut être résolu en utilisant un solveur MILP comme Gurobi.

Comme un fournisseur de services peut souhaiter exécuter fréquemment la mise à jour du modèle, il peut être intéressant de fournir des algorithmes plus rapides qui s'exécutent en temps polynomial. Nous proposons donc également des heuristiques basées sur des algorithmes conçus pour les problèmes de *Set Cover* ou de *Weighted Set Cover*. Dans ce cas, nous modélisons chaque nœud

Fog comme l'ensemble de stations de base auxquelles ce dernier est connecté tout en respectant les contraintes de qualité de service. Nous essayons de minimiser le nombre des nœuds Fog à sélectionner pour couvrir toutes les stations de base. L'ajout des poids permet d'obtenir la solution la plus optimale en fonction de la quantité de trafic qui est envoyée entre les stations de base pour des régions plus actives du trafic.

Nos expériences montrent que pour de petites instances le solveur et les heuristiques donnent des résultats similaires. Pour des instances de plus grandes tailles, Gurobi obtient des solutions significativement plus efficaces. En terme de temps d'exécution les algorithmes gloutons sont bien sûr plus performants. Il s'agit donc d'un compromis à déterminer en fonction du temps d'exécutionet du niveau de sous-optimalité acceptables.

Le modèle que l'on propose dans cette thèse est une première approche simple pour l'optimisation des migrations de services dans le cas d'utilisateurs mobiles. Malheureusement il ne traite pas correctement un certain nombre de scénarios. Par exemple, tandis qu'ajouter plus de nœuds Fog devrait naturellement conduire à plus de migrations, ce modèle les annule ce qui entraîne une diminution des coûts. Une approche qui reflète mieux la réalité basée sur des chemins résout ces problèmes et est proposée comme perspective dans cette thèse. Cependant, si l'on n'a que les statistiques de hand-over sans pouvoir suivre les utilisateurs, ce modèle fournit une solution qui encourage la minimisation de migrations.

Troisième Problématique

Dans la dernière partie de la thèse, nous nous intéressons à l'allocation dynamique des tâches dans le réseau Fog. La question à laquelle nous cherchons à répondre est la suivante: comment une station de base doit-elle répartir les tâches qu'elle reçoit entre les différentes stations de base ? Nous nous intéressons à des algorithmes permettant d'adapter dynamiquement l'allocation de tâches de manière à pouvoir réagir à différentes perturbations ou événements adverses (variations de trafic, pannes, etc). Nous nous focalisons sur une approche entièrement décentralisée dans laquelle les stations de base n'ont aucune information sur l'infrastructure et ne se coordonnent pas.

L'approche que nous suivons est expérimentale: nous proposons un modèle de simulation pour observer et comparer différentes stratégies et leur robustesse sous différentes conditions. Nous utilisons pour cela le framework OMNET++ pour créer une simulation et comparer les résultats de ces algorithmes.

Les nœuds Fog et Cloud sont modélisés comme des ensembles de files d'attente parallèles à n serveurs. De nombreux paramètres (nombre de serveurs, distribu-

tion des temps de service, taux d'arrivée, latence de communication etc.) sont ajustables et permettent de définir différents scénarios. Par exemple, nous utilisons une distribution exponentielle pour simuler les conditions normales de trafic et une distribution de Pareto pour simuler des conditions avec plus de variabilité. Le modèle de simulation permet par ailleurs l'injection de perturbations (des variations de trafic par exemple) afin d'étudier à quelle vitesse les algorithmes d'allocation de tâches sont capables de s'y adapter.

Les différents algorithmes d'allocation de tâches que nous simulons n'ont aucune information sur l'infrastructure de calcul. Ils réagissent uniquement aux temps de réponse qu'ils observent des différents micro-datacenters. Nous considérons des algorithmes d'apprentissage très simples, comme l'algorithme ϵ-*greedy*, l'*allocation Softmax*, l'*algorithme EXP3* développé pour les bandits manchots adversariaux ou encore l'approche *Sensible Routing* proposée dans [Gelenbe 2003, Wang 2018a]. Les performances de ces algorithmes (temps de réponse moyen obtenu, vitesse de convergence suite à une perturbation, robustesse à la variabilité du trafic, etc.) sont comparées entre elles, ainsi qu'avec une allocation statique dans laquelle les tâches sont systématiquement routées vers le nœud Fog le plus proche.

Nous étudions sept scenarii différents. Pour tous ces scenarii, la méthode *Sensible Routing* est celle qui offre les meilleurs temps de réponse. On remarque toutefois que l'*algorithme EXP3* semble plus plus robuste à la variabilité des temps de service. On note par ailleurs que EXP3 réagit mal très temporairement au moment des perturbations.

Dans trois de ces sept scénarios, nous nous intéressons à une extension du modèle avec de l'offloading de tâches. Dans ce modèle les nœuds Fog sont connectés au Cloud et ils ont la possibilité d'y envoyer leurs tâches. Cette décision est prise localement par le noeud Fog qui estime le temps de traitement dans le Cloud. Le noeud Fog compare ensuite ce temps de traitement dans le Cloud avec la durée moyenne d'un traitement local. Finalement, le nœud Fog décide de transmettre la tâche au Cloud si la durée de traitement dans le Cloud est inférieure.

Dans ces trois scénarios exécutés avec l'offloading, nous remarquons que Sensible Routing fournit les meilleures performances. En comparant Sensible Routing avec et sans offloading, nous remarquons que l'offloading va systématiquement améliorer les temps de réponse au moment de perturbations. On constate aussi que c'est la combinaison d'une allocation de tâche adaptative et d'un mécanisme d'offloading qui permet d'obtenir les meilleures performances. Bien que les stations de base aient la possibilité d'envoyer directement leur tâches au Cloud, elles préfèrent les envoyer aux nœuds Fog qui ont la possibilité d'offloader.

Enfin, tous les services ne sont pas critiques. Pour certains services critiques il n'est sûrement pas judicieux de faire de l'offloading. Il serait donc possible de conserver de la capacité de garantir un traitement local dans le Fog.

Contents

List of Figures

List of Tables

Introduction

While mobile networks were designed first and foremost to support voice, SMS messages quickly became the new unit of communication. With the advent of smartphones, data became prominent. As smartphones got more computationally powerful and were made programmable by third party application developers, the Cloud became the de facto hosting platform for the processing and the storage of applications.

The Internet of Things (IoT) will connect various objects, buildings or facilities to the Internet and will thus result in another dramatic increase of network usage. Among these new IoT applications, *connected cars* will be pervasive: according to the AECC, the Automotive Edge Computing Consortium [AEC], by 2025 the number of connected vehicles will grow to around 100M globally and the data volume transmitted between vehicles and the Cloud will be around 100 petabytes per month.

This breed of applications will likely contain latency critical features that have stringent delay requirements that will not be satisfied by the far away Cloud. This calls for the extension of the classical centralized Cloud computing architecture towards a distributed architecture closer to user premises at the *edge* of the network. Different extensions were proposed over the last decade as new paradigms such as *Edge Computing*, *Fog Computing*, *Cloudlet*, *Multi-Access Edge Computing*, *Mist Computing*...

Through this picture we see we are moving to a world where network and computing resources will become another ubiquitous utility like water and electricity. Just like any other utility, network and computing is serviced by providers who aim to minimize their costs.

One way for Cloud operators to reduce cost is to share hardware and infrastructure between many clients. This is made possible notably by using *Virtualization*. Virtualization is a proven technique in software that gives the illusion of having multiple machines executing in one physical machine, allowing different clients to share resources in an isolated manner. Another key benefit of virtualization is the possibility of migrating virtual machines from one computer to

another. This helps to redistribute computing load and keep services up in case of hardware failure. This is called *VM migration*.

This thesis focuses on techniques useful to minimize infrastructure cost for Fog providers in the specific context of connected cars. Many stakeholders can be behind this infrastructure: telecommunication providers, Cloud providers, the automotive industry, etc.

As a part of our work we have studied the problem of minimizing the cost of a newly designed infrastructure for Fog computing and the associated routing strategy as a function of *compute capacity* and others factors such as *operational*, *installation* and *energy* costs, considering workload characteristics as *statistics of daily patterns* and *Quality-of-Service*.

Another challenge for connected cars is the proper support of mobility for Fog applications. Modern applications are made of a client and a server side called service. Fog computing plans to support service migrations in order to keep the low latency requirements as the user moves. However frequent migrations introduce an overhead. In our second study, we aim to minimize the costs associated with service migrations triggered by user mobility.

While a number of decisions can be taken at the design phase, these decisions are taken based on estimations. Later, in the operation phase, we need to handle real patterns that do not entirely match the estimations and we need to employ methods that adapt to actual traffic. In our last study we propose a simple simulation model for Fog computing based on OMNeT++ to compare the performance of a number of state-of-the-art task allocation algorithms. We focus on model-agnostic task allocation algorithms which only react to the response times they observe from Fog and Cloud nodes.

Thesis Organization

This thesis is structured in the following way:

- Chapter 1 provides an overview of Cloud Computing and other associated paradigms that take computations to the *"edge"* of the network. We particularly focus on *Fog Computing* which provides a continuum between the Edge and the Cloud. An overview of requirements specific to *connected cars* is presented.

- Chapter 2 presents the mathematical background and the techniques we have used for this thesis, most notably *Optimization* and *Queuing Theory*.

- In Chapter 3, we introduce the problem of *optimal capacity planning* for a newly designed Fog Infrastructure. We present our mathematical formula-

tion, its resolution as a mixed-integer-linear programming formulation, and our results.

- In Chapter 4, we examine ways to minimize overhead associated with service migrations in the Fog. We focus mainly on an approach that takes handover events as an input and suggest a more effective solution that requires path based mobility patterns.
- In Chapter 5 we present a simulation model as well as the necessary background on the OMNeT++ simulator. We present a number of adaptive task allocation methods. We provide comparisons and performance tests. We test the algorithms in varied conditions in order to observe how robust they are.
- In Chapter 6 we present a discussion on Fog computing, a summary of our research contributions and future research directions.

Happy reading!

Chapter 1

Cloud and Edge Computing Paradigms

With the advent of the global Internet, network infrastructure has evolved to serve computing. The major distant computing paradigm now in use is the Cloud. Cloud computing is a centralized paradigm that provides a shared computing, networking and storage infrastructure for computer applications.

The Cloud solves major concerns allowing economies of scale but is not suited to all applications. Notably, in sharp contrast to traditional cloud services, many emerging applications of the *Internet-of-Things* (IoT) require low and predictable latency. To support these use cases, extensions of the classical centralized cloud computing architecture have been proposed, bringing a distributed architecture closer to user premises at the *edge* of the network.

Different extensions were proposed over the last decade as new paradigms such as *Edge Computing*, *Fog Computing*, *Cloudlet*, *Multi-Access Edge Computing*, *Mist Computing* pushing computing to the edge. All these extensions take computing closer to the user at the *edge* of the service provider's network[1].

In this chapter, we first present the cloud and edge computing paradigms. Then we introduce the different distributed computing paradigms proposed in the literature. We focus in particular on the Fog computing paradigm and its application to connected cars.

[1]Because it depends on the point of view of the service provider, the actual position and definition of the edge in the literature vary.

1.1 Cloud Computing

Cloud computing is a centralized paradigm that provides a shared computing, networking and storage infrastructure for computer applications accessed throughout the Internet.

In the early days of computing, hardware came under the form of large computers known as *mainframes*. Because computational power was very expensive, users would connect to the mainframe using "dumb" terminals[2]. This approach is called *time sharing*: instead of running one process or computation at a time the operating system is able to run several which share CPU time. This technique led to the design of multi-users systems where users could all run their own processes, offering an opportunity for better sharing of resources. At a given time some users may be actively running processes while others are idle [Tanenbaum 2015].

With the advent of the Internet and with the increase of network bandwidth, it has become possible again to use a similar model of shared computing resources, this time across long distances. A number of Internet actors used this opportunity to introduce a centralized paradigm as a cost-effective solution.

This approach was called *Cloud Computing* because the amount of resources and the precise underlying architecture of the system is hidden from the users.

Cloud Computing is championed by companies such as Amazon, Google, IBM, Microsoft. There are three main kinds of offering : *Infrastructure-as-a-Service* (IaaS), *Software-as-a-Service* (SaaS), *Platform-as-a-Service* (PaaS) and more recently *Function-as-a-Service* (FaaS).

"Infrastructure-as-a-Service" lets companies rent servers running Linux or Windows. They can pick different hardware configurations for dedicated or *virtualized* resources. It is possible to programmatically create or destroy new server instances through *APIs*[3]. This lets companies adjust the amount of hardware they need to handle the load dynamically. For instance a company may add more servers on peak day or it could be automatically scaled by an orchestrator. Key IaaS services are *Amazon AWS EC2*, *Google Compute Engine* or *Microsoft Azure*.

"Platform-as-a-Service" provides a complete software platform rather than a bare operating system. For instance it may provide a Java Entreprise environment where companies can deploy their applications without necessarily controlling the operating system configuration or directly addressing scale and load concerns. Some PaaS platforms will automatically adjust the required resources to run the

[2] "Dumb" terminals let users run jobs on the mainframe, but do not have dedicated computing resources.

[3] Application Programming Interface

application under a certain quality of service. Some examples are *Google App Engine*, *AWS Elastic Beanstalk*, *Heroku*.

"Software-as-a-Service" provides companies with readily running software that is already configured and will scale on-demand. Examples of SaaS include *AWS SQS*[4] which is a queue service hosted and run by Amazon, or *Elastic Cloud* which is a hosted data storage service. A lot of software is now distributed as SaaS instead of running on the companies infrastructure because it minimizes the maintenance cost of running the software.

"Function-as-a-Service" is a simpler form of PaaS. Instead of providing a complete platform to build Cloud applications against, it lets users define simple functions that are wired together by the Cloud provider, for instance when an HTTP request is made. *AWS Lambda* is a well-known FaaS service.

More and more companies deploy their application to the Cloud to benefit from the cost savings and simplify their systems. However, the centralized model of the Cloud comes with downsides. Datacenters might be installed far from the users which may incur higher latency. Emerging IoT applications may not be able to meet their delay requirements.

Fog Computing and other edge computing paradigms are extensions of the core idea of the Cloud and address these latency requirements by distributing and sharing computing and networking power at the edge of the network. Fog Computing was introduced by Cisco as a complement of the Cloud to reach the edge of and extension of computing to the edge of the network. These paradigms will be presented in the following sections but we will first introduce *virtualization* and *VM migrations* which are two key techniques used in the Cloud and in other distributed computing environments.

1.1.1 Virtualization

Virtualization is a technique that allows software to emulate hardware in order to "create" *virtual* machines (VM). This technique lets multiple isolated operating systems to run on the same hardware at the same time. The software that handles the interactions between multiple virtual machines and the hardware is called the *hypervisor*.

Because virtual machines contain a full operating system and hardware emulation, they imply a lot of overhead. A lighter, more recent technique known as *containerization* provides some of the same benefits as virtual machines while avoiding hardware emulation: in this case the operating system itself provides the functionality to isolate executable code [Zhang 2018b].

[4] Simple Queue Service

Both VMs and containers present a standard interface that allows them to run on different machines and to be deployed for fault-tolerance and load-balancing. In general both techniques allow for economies of scale by ensuring the hardware is well utilized thus minimizing hardware and electricity cost [Tanenbaum 2015].

1.1.2 VM migration

As discussed in the section about Virtualization, virtual machines are decoupled from physical servers. VMs can be used for fault-tolerance where the system can be resumed in case of hardware failure by just copying and running the affected images. Last, VMs can help load balancing under certain conditions such as the statelessness of the provided services [Tanenbaum 2015].

The event of transferring a VM from one server to another is called a *VM migration*. There are two main approaches to migrate, one is to shut down the virtual machine and migrate it and the other is to pause the virtual machine and migrate it. The former is called *cold migration* and the latter *live migration* [Zhang 2018a]. The precise mechanism of these functions is outside the spectrum of this thesis but we should mention that migrations introduce an overhead and there is a *migration time* that is equal to as the total time to transmit and resume the VM [Li 2019].

In the problem we are presenting in Chapter 4 we are aiming to minimize the number of migrations that will take place in distributed architectures such as the Fog.

In Cloud Computing, migrations are useful to maintain higher system utilization and to lower energy consumption [Bittencourt 2015] by sharing hardware resources. One extra challenge in Fog Computing is the mobility of the client.

1.2 Edge Computing Paradigms

1.2.1 Edge Computing

Edge computing is a paradigm aiming to move the computation and the storage resources at the edge of the network.

The position of the edge depends on the service provider. For instance if the edge service is provided by the Internet Service Provider (ISP), the edge server is the first hop on the operator infrastructure from the user's point of view. If the edge provider is another actor the edge will be further in the Internet but still expected to be closer to the user than a centralised Cloud solution [Fas].

One goal of Edge computing is to minimize the usage of network resources by keeping the computation closer to the user and thus not requiring packets to travel through many hops. Another benefit of the Edge is to keep latency low, making it possible to offer a good Quality of Service (QoS) to latency-critical applications.

Edge computing is a generalization of an older concept: *Content Delivery Networks* (CDN) [Pathan 2012]. CDNs allow the distribution of static content like Web files and videos from a number of *Points of Presence* (PoPs) that are distributed geographically around the world. Because static files may be very large, distributing them from a centralised location would cause network congestion and could be a bottleneck. CDNs provided a solution to serving the same content to millions of users at the same time. Major CDN providers include Akamai, CloudFare, Fastly, Limelight Networks etc.

Edge computing takes the concepts of CDN further by applying the same model to general computation instead of just static content.

1.2.2 Fog computing

While the Cloud is about sharing infrastructure and finding cost saving opportunities, the Edge is about distributing infrastructure closer to user in order to increase availability and improve *Quality of Service* (QoS).

They are complementary and allow a better usage of resources. The Cloud does it by sharing computing resources while the Edge computing does it by keeping computation local, making it useless to reach the deeper Internet and saving network resources.

Fog Computing recognizes the benefits of both the Cloud and the Edge and tries to bridge them seamlessly in a unified model where different jobs can be processed on either the Edge or the Cloud or migrated from one to the other depending on their requirements. The cooperation between the Cloud and the Fog is handled by a Fog orchestrator which can control the QoS and then can deploy and schedule resources to the Cloud or the Fog appropriately.

The new paradigm was introduced by Cisco in 2015 [CIS] and since then the vision of Fog has been pushed forward by the *OpenFog* consortium [OPE]. The OpenFog consortium, consists of industry companies and academic institutions that cooperate in order to promote and standardize it.

In addition, the American National Institute of Standards and Technology-NIST has provided a formal definition [NIS].

> *Fog computing is a horizontal, physical or virtual resource paradigm that resides between smart end-devices and traditional cloud or data*

centers. This paradigm supports vertically-isolated, latency-sensitive applications by providing ubiquitous, scalable, layered, federated, and distributed computing, storage, and network connectivity.

A Fog Computing architecture is a highly virtualized platform that provides a multitude of compute, storage, and networking resources at the edge of the network, allowing applications that depend on time-critical data to use nodes in their vicinity to meet the delay requirements [Mims 2014, Bonomi 2011, Bonomi 2012, NTT 2014], [NTT 2014]

The architecture of Fog is illustrated in Figure 1.1.

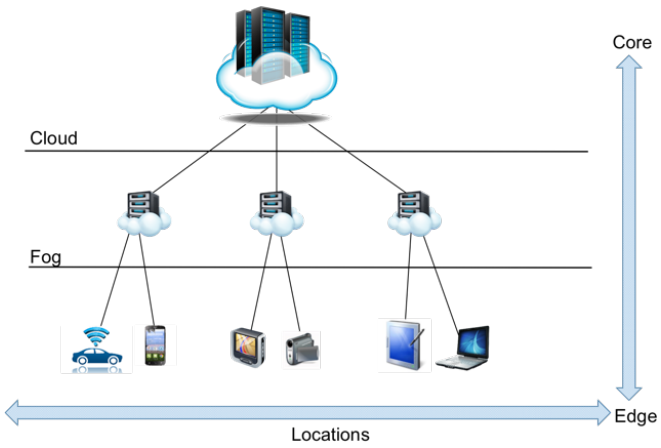


Figure 1.1: Fog Computing

At the bottom of the diagram we have the IoT devices which can be a number of applications that can benefit from the Fog as transportation IoT applications, agriculture, smart cities, healthcare etc. In the middle layer there is the Fog. which consists of the *Fog nodes* that are geographically distributed. As pointed out in Section 1.1 these resources can be routers, switches, gateways, set-top boxes or access point and servers in micro data centers which can provide the Fog compute. NIST names all of these *Fog nodes*. However in the literature you can find terms such as *microdatacenter*, *microcloud* or *cloudlet*. Finally, at the top of the figure we have the Cloud which works in cooperation with the Fog.

Unlike Edge computing, Fog computing does not focus exclusively on the edge, but on the continuum of service from the Cloud through the Edge down to IoT devices.

There are a number of papers in the literature that interchangeably use the terms Fog and Edge but the NIST and the OpenFog consortium explicitly distinguish the two with edge being just a component of the Fog.

1.2.3 Mist Computing

Mist computing [Yousefpour 2018] is another paradigm which has been proposed to approach computing happening at the extreme edge: the IoT devices themselves. It can cooperate with the Fog and the Cloud as defined by the European Telecommunications Standards Institute (ETSI).

1.2.4 Cloudlet computing

Cloudlets are small data centers that are typically one hop away from the mobile devices. The term was first coined by Satyanarayanan, Victor Bahl, Ramón Cáceres, and Nigel Davies and a prototype version was developed then by Carnegie Mellon University [Yousefpour 2018, Ceselli 2015][Dolui 2017]. The Fog network can also make use of cloudlets as micro-datacenters and they are part of the Fog nodes. In our studies we consider Fog nodes that are cloudlets as they have their computing storage resources independent from the network hardware.

1.2.5 Multi-Access Edge Computing

Another well studied computing paradigm in the literature is the *Multi-access Edge computing* [Tanaka 2018][Porambage 2018] [Pham 2019]. It was previously referred as mobile edge computing but the term multi-access includes applications other than mobile device-specific tasks.

In MEC, Cloud computing capabilities are provided in existing base stations. It is taking advantage of the cellular infrastructure and it is expected to benefit from 5G. It is a paradigm standardised in ETSI [MEC].

1.3 Fog Computing for Connected Cars

The main motivation for our work comes from connected vehicles for which a large number of applications with varying needs of quality of service have been defined. Applications for connected vehicles can be classified into three categories [ets 2009]:

1. Active road safety applications: applications used to reduce the probability of traffic accidents and loss of lives. Examples include: intersection collision warning, head on collision warning, emergency vehicle warning, wrong way driving warning, signal violation warning, etc.

2. Traffic efficiency and management applications: these applications are employed for improving the traffic flow, traffic assistance, traffic coordination, updated local information, etc. Example include speed management, and cooperative navigation applications.

3. Infotainment applications: these are less constrained applications such as application that collect and disseminate information about locally based services such as points of interest (restaurants, hotels, etc.) or global internet services: multimedia services, parking management, etc.

Multiple actors may be involved in the building and operating of infrastructure at the *"edge"* of the network. We can mention for instance: *Telco providers*, *car manufacturers*, *road infrastructure operators* or *Cloud providers*. Depending on the requirements of each provider, computational capacity can be assigned at different parts of the network infrastructure and thus different services can be offered.

In the particular case of connected cars there are critical *safety* requirements where low latency is expected. Based on [ets 2009], these applications have different needs, ranging from the most constrained: periodic messages, $10Hz$ frequency and $100ms$ critical latency for active road safety applications to $1Hz$ frequency and $500ms$ critical latency for co-operative services.

It is thus expected to be able to carry these computations in the immediate vicinity of the user. One vision that can be found in the literature is that processing nodes are on-board the connected cars themselves. In the research literature this paradigm is known as *Vehicular Fog Computing* [Xiao 2017a] [VFC].

There are two potential schemes for Connected Vehicles: *Dedicated short-range communications* (DSRC) and Cellular Networks [Xu 2017] The first one can be used in scenarios where Road Side Units (RSUs) are installed on the roads and be then used as processing nodes. Although RSUs can enlarge the network communication capacity, they are really expensive and are difficult to be fully deployed along roads, on a large scale [Veh]. In a second approach we have communication using the cellular network. *Base stations* were first used as a means to support mobile telephony but since they have started handling data. As we saw before *Multi-Access Edge Computing* aims to take advantage of the existing base stations and add to them virtualized cloud computing capability.

The edge of the network from the point of view of a network provider are the devices that exist in the network: routers, switches, gateways, host machines.

In our studies, the compute power provided by the Fog is inside the micro-data centers that will be installed at the edge of the network near the base stations.

Chapter 2

Optimization and Queuing Theory

This chapter provides an overview of the mathematical theory used in this thesis. In Chapters 3 and 4, we address problems related to the optimal design of geo-distributed computing infrastructures.

In Chapter 3, we consider the Fog provider is in the infrastructure design phase and we are studying the optimal trade-off between a centralized and distributed solution in terms of workload characteristics and compute capacity. In Chapter 4, we consider the Fog node locations are fixed and we are studying ways to consume less networking bandwidth and compute resources by minimizing the number of service migrations due to connected car mobility.

These are problems addressed by *Optimization* techniques. We present the central techniques applicable to our works in this chapter.

We also use *Queuing Theory* in our models to express the probabilistic delay requirements for the quality of service in the Fog. Queuing Theory is part of the larger field of *Stochastic Modelling and Analysis* that is applied to the study of queues that appear for instance in a *router*, in a *server* or as well in lines in a supermarket counter. It is often necessary to understand the relationship between adding more resources and performance. This is a known application of queuing theory named *capacity planning or capacity provisioning*. It is very useful in designing networks and data centers. We further present the necessary background in this chapter.

2.1 Optimization

An optimization problem has the following form:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq b_i, \quad i = 1, \ldots, m \\ & h_j(x) = 0, \quad j = 1, \ldots, m \\ & x \in \mathbb{R}^n \end{array} \tag{2.1}$$

In this formulation, the function f is called the *objective function*. Each of the g_i is an inequality constraint and each of h_j an equality constraint. A vector $\mathbf{x} \in X$ which satisfies all of the constraints is called a *feasible solution*.

The goal is to minimize the function $f(x)$ under equality and inequality constraints by searching for a feasible solution x^* that minimizes the value of the objective function f, $f(x^*) \leq f(x)$. It is also possible to find the values x that maximize the value of the function $f(x)$ by replacing the function $f(x)$ by the function $-f(x)$.

There are two main axes that help categorize optimization problems: whether the objective function and associated constraints are linear or nonlinear, in which case there are subcategories depending on the shape of the function, and whether the variables are continuous, discrete or a mix of both.

If the function and constraints are linear and some of the variables are discrete and some continuous we fall into the category of *Mixed Integer Linear Programming*. When the objective function and the constraints are linear and the variables can only take integer values we fall into *Integer Linear Programming* problems.

When the function or the constraints are nonlinear, we fall into the category of *nonlinear programming*. There is no global optimum in that case, but several local optima. If the objective function and the constraints are convex or concave then we fall into the category of *Convex Optimization* [Boyd 2004] where the local optimum is the global.

2.2 Linear Programming

This category of optimization contains formulations where the objective and the constraints are linear functions. Linear programming is split in different categories depending on the type of the variables: real numbers $\mathbb{R}$, integer numbers $\mathbb{Z}$ and mixed problems where some variables belong to $\mathbb{R}$ and some variables belong to $\mathbb{Z}$.

Linear programming is a category of optimization where the objective function is linear and subject to linear equality and inequality constraints [Bertsimas 1997]. The set of feasible solutions is a polyhedron belonging to $\mathbb{R}^n$.

Two important algorithms used to solve linear problems are the *Simplex method* and the *Interior Points method*.

The Simplex method was proposed by George Dantzig in 1947 to help military planning during World War II and has been used since then in a number of fields such as transportation, economics or scheduling.

The core idea of the Simplex method is to navigate the external edges of the polyhedron that underlines the linear program and always move to a more optimal vertex until the optimum is reached. In the worst-case the algorithm performs in exponential time but the average-case complexity is polynomial. Figure 2.1 illustrates the simplex search.

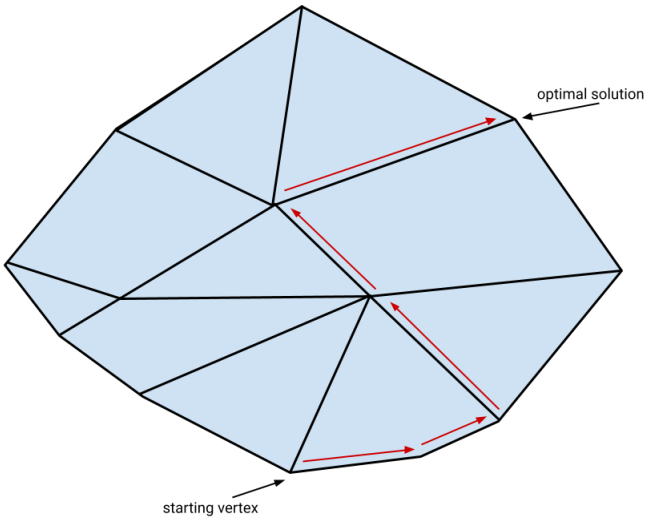


Figure 2.1: The Simplex search

It is possible to avoid the exponential worst case by using a different strategy. This is what the *Interior Points* (also known as *Barrier*) method achieves by navigating the inner vertices through the polyhedron. Interior point methods have a long history but they became popular thanks to Karmarkar's works in 1984 [Strang 2005]. The algorithm runs in polynomial time and it is very efficient in practice.

These two algorithms are commonly implemented by linear programming solvers. There are variations in the implementation thanks to research and tech-

nical progress that allow some implementations to perform better than others [Gurobi].

2.3 Discrete Optimization

Discrete optimization focuses on problems where variables belong to a discrete set such as the natural numbers $\mathbb{N}$ (Integer Programming) or a subset of it such as $\{0, 1\}$ (Binary Programming). Intuitively, one could think that this is an easier category of problems to solve since the set of feasible values is smaller, compared to analogous continuous optimisation problems. However this category of problems is much more difficult because a number of approximation tools are not available on discrete problems.

There are three types of algorithms solving discrete optimization problems [Bertsimas 1997].

1. *Exact algorithms*: they provide an exact solution but sometimes imply an exponential complexity that is intractable in practice. Exhaustive search is a brute force approach which enumerates all solutions and triggers a combinatorial explosion that makes unsuitable to solving real-world problems. Other methods that are much more efficient include *Branch-and-Bound*, *Branch-and-Cut*, *Cutting Plane* and *Dynamic Programming*.

2. *Approximation algorithms*: they return an approximation that is guaranteed to be within a certain ratio of the optimal value. They usually perform much better as they sacrifice exactness for speed. In practice these algorithms usually run in polynomial times. We present an example of such an approximate solution for the *Set Cover* problem below.

3. *Heuristic algorithms*: they return an approximation without guarantee of being within a ratio to the optimal value. They work best on certain instances of a given problem and their pathological cases may return solutions very far from the optimal. Examples include *Local Search methods* and *Simulated Annealing*.

Branch-and-Bound

Branch-and-Bound is a fundamental technique behind Integer Linear Programming. It is a family of algorithms that produce exact solutions to linear discrete optimization problems. It was developed in 1960 by Land, Doig and Dakin.

The algorithm explores possible solutions by storing partial solutions in a tree structure. The tree traversal is done by following *Breadth-First-Search*. Each node of the exploration tree contains the current decision for some of the problem's variables. The tree is lazily constructed by creating new branches (*Branching*), and by exploring only sub-trees that are guaranteed to contain the optimum: this is achieved by the *Bounding* phase that excludes sub-trees that are known to be above or under a certain threshold analytically computed for the problem at hand [Wolsey 1998]

Commercial solvers combine Branch-and-Bound with other elaborated techniques such as cutting planes and heuristics to achieve better performance [Gurobi].

2.3.1 Integer Programming and Combinatorial Optimization

An important category of Integer Programs is the *Combinatorial problems* [Levitin 2002]. Their feasible solutions are combinatorial objects such as permutations, combinations or subsets which satisfy constraints. Most of them are difficult to solve with exact algorithms as the number of combinatorial objects grows with the problem size. The *Set Cover* problem is a family of combinatorial problems that is applicable to many real-world situations. It has no efficient exact algorithm but fast and good approximations which we use in the problem we study in Chapter 4.

Set Cover Problem

The Weighted Set Cover Problem is a known problem in the area of combinatorial optimisation which is NP-complete. It is a generalization of the Set Cover problem which is specified by a set of elements $\{1, 2, ..., n\}$ called the *universe* and a collection S of m sets whose union equals the universe. The Set Cover problem is about identifying the smallest sub-collection of S whose union equals the universe. Figure 2.2 illustrates an instance of it.

This problem has many practical applications, e.g airline scheduling, vehicle routing, facility location. No polynomial time exact algorithm exists, so a greedy approach can be used as an approximation algorithm. In the greedy strategy, we will iteratively pick the biggest subset that contains the most number of uncovered elements.

In the weighted version of the Set Cover problem a cost function is associated to each subset. The objective is to find the subset that has the least total cost.

Problem 1 (Weighted Set Cover) *Given a universe U of n elements, a collection of subsets of U, $S = S_1, \ldots, S_k$, and a cost function $c : \mathcal{S} \to \mathbb{R}_+$, find a*

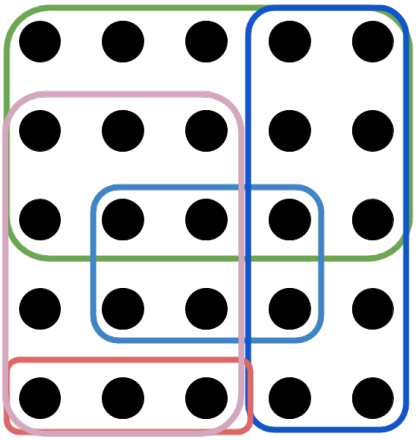


Figure 2.2: Set Covering problem

minimum cost sub-collection of $\mathcal{S}$ that covers all elements of U [Vazirani 2001].

In the greedy strategy for the weighted version, instead of picking the subset that adds the subset that has the most number of new elements, we iteratively pick the subset with the smallest $\frac{c(S)}{|S \setminus C|}$.

The greedy weighted set covering heuristic does the following [Vazirani 2001]:

Algorithm 1: Greedy Weighted Set Cover algorithm

```
C ← ∅
while C ≠ U do
    Pick S with the smallest c(S)/|S\C|
    C ← C ∪ S
Output the picked sets.
```

2.3.2 Mixed Integer Linear Programming

Mixed Integer Linear Programming is a branch of discrete optimization where variables can take continuous and integer values. Two well-known examples of problems that can be represented as MILP problems are the *Knapsack* and the *Facility Location* problems. Facility Location is a family of problems common in Operational Research that focuses on the optimal placement of facilities to minimize transportation and installation costs and serves clients economically.

The problem we address in Chapter 3 deals with the optimal placement of micro-data centers, the *facilities*, in terms of installation and server compute cost by allocating optimally the connected car demands to datacenters.

Facility Location

The core idea of facility location is to select a subset of potential locations $N = \{1, \ldots, n\}$ to place facilities and assign a set of client demands $I = \{1, \ldots, m\}$ to these. The objective is to minimize total cost, given the cost of operating the facility i is f_j and c_{ij} the cost of serving the demand of a user i from facility j.

When there is no bound on the number of clients that a facility can serve we talk about uncapacitated facility location problems whereas we talk about capacitated facility location when there is restriction on the number of clients. The restriction on capacities is modeled by u_i. In the following models, the variable x_j is a binary variable representing whether the facility will be opened or not [Marzie Zarinbal (auth.) 2009] [Michele Conforti 2014].

Capacitated facility location problem

In the capacitated case there is a demand d_i and a fraction of the demand noted as y_{ij}.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} d_i y_{ij} + \sum_{i=1}^{n} f_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} y_{ij} = 1 && \text{for all } i = 1, \ldots, m \\
& \sum_{i=1}^{m} d_j y_{ij} \leq u_i x_i && \text{for all } j = 1, \ldots, n \\
& y_{ij} \geq 0 && \text{for all } i = 1, \ldots, n \text{ and } j = 1, \ldots, m \\
& x_i \in \{0, 1\}
\end{aligned}
\tag{2.2}
$$

Uncapacitated facility location problem

When there is no restriction on the capacity of the facility, y_{ij} can be assumed binary and replaced by the variable z_{ij} which is 1 when client i is served by facility

j

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij} d_j z_{ij} + \sum_{i=1}^{n} f_i x_i & & \\
\text{subject to} \quad & \sum_{i=1}^{n} z_{ij} = 1 & & \text{for all } j = 1, \ldots, m \\
& \sum_{j=1}^{m} z_{ij} \leq M x_i & & \text{for all } i = 1, \ldots, n \\
& z_{ij} \in \{0, 1\} & & \text{for all } i = 1, \ldots, n \text{ and } j = 1, \ldots, m \\
& x_i \in \{0, 1\} & & \text{for all } i = 1, \ldots, n
\end{aligned}
\tag{2.3}
$$

This is the general modeling of a main category of Facility Location problems; however our description is richer and differs since we are taking into consideration optimal allocation of the tasks with respect to quality of service delays and the cost of compute capacity to allocate to datacenters. These quality of service delays are introduced in the form of probabilistic constraints.

2.4 Non-linear Optimization

Nonlinear optimization studies problems where the objective or constraints functions are not linear. This category of problems can be complex and it is difficult to find a global optimum value. Instead, a compromise can be to find several local optima. If there are no constraints to the optimization problem, *gradient descent* based methods can be used to find local optima fast [Boyd 2004].

Gradient methods

Gradient descent methods as the name indicates, are based on the descent direction of the gradient that give the direction to to the minimal value.

The general idea is that they produce a sequence

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} \Delta x^{(k)} \tag{2.4}$$

where Δx is the opposite direction of the gradient that decreases the cost function 2.7:

$$\Delta = -\nabla f(x^{(k)}) \tag{2.5}$$

and when it is applied on the vector x results in

$$f(x^{(k+1)}) < f(x^{(k)}) \tag{2.6}$$

.

The scalar α is called the *step size* or the *step length* at iteration k.

The optimum step size is computed by solving the following one dimensional problem:

$$f(x^{(k)} + \alpha^{(k)}\Delta^{(k)}) = \min_{\alpha \in \mathbb{R}} f(x^{(k)} + \alpha\Delta^{(k)}) \tag{2.7}$$

Chance Constrained Optimization

The previous optimization techniques presented so far required deterministic formulations where all the input is known in advance. When the input contains random variables we fall into the optimization under uncertainties. One category is called Chance Constrained Optimization.

The formulation of such problem can contain a constraint in the form a probability as in the following:

$$\begin{aligned} &\text{minimize} && f(x, \xi) \\ &\text{subject to} && \mathbb{P}(h(x, \xi) \geq 0) \geq p \text{ where } p \in [0, 1] \end{aligned} \tag{2.8}$$

where ξ is the vector of uncertainty.

When these problems arise we need to have an intuition of the problem domain in order to make some assumptions.

The problem we are presenting in Chapter 3 deals with the optimal capacity planning of a Fog Computing infrastructure under *probabilistic delay guarantees*. In order to find the optimal capacity planning we take into account fixed networking delays and a processing delay which is modeled as a random variable. In our case the processing delay depends on the load of the server. However, using Queuing Theory, we make assumptions on the statistical distribution that the processing delays follow and we can analytically transform the problem into a MILP formulation. Queuing theory is presented in the next section.

2.5 Queuing Theory

Queuing theory is an area of mathematics dedicated to, the study of "queues". Queues are pervasive in systems whether they are information systems or real

life systems such as supermarket counters, highway toll stations, airport security checks and so on.

By modeling the system using Queuing Theory, we can *dimension* it better and study its average performance.

Examples of scenarios applicable to queuing theory are:

- How many checkout counters are needed in the supermarket in order for clients to tolerate waiting in the line to pay?
- How many lanes should be built on a highway to keep the traffic fluid?
- What is preferable to have: a single powerful server or several less powerful servers ?

In computer systems, queuing theory is used to model situations such as concurrent access to a server, a disk, memory, bandwidth.

Our problem in Chapter 3 is of the same nature: How many compute servers should we allocate in each Fog node in order to obtain a minimum cost infrastructure under probabilistic delay guarantees ?

Queuing Theory was developed around the 1900s by Agner Krarup Erlang who modeled the telephone network in Copenhagen.

The research applying to the telephone network has evolved since into being one of the most important tools for the study of telecommunications and of the Internet. The field now has a variety of applications such as the design of routing protocols for networks, of better scheduling algorithms for Web servers and database management systems, disk scheduling, power management and capacity provisioning in data centers, etc.

In order to specify a queuing system we need to model the following: the arrival stream, the service duration, the number of servers, the structure of the service facility, the maximum of number of jobs in the system and the discipline of the service facility, etc[Adan 2001]. We present these concepts in more details in the following sections.

Arrival process

The evolution over time of the number of jobs in the system is modeled by a random process and can be described in two ways:

- by the average number of arrivals per unit time known as the *arrival rate* and noted as λ

- by characterizing the time between two successive arrivals: this is also called the *inter-arrival time*

Since in simple models it is assumed that consecutive inter-arrivals are independent and they follow the same probability distribution, the arrival process is defined using the probability distribution of inter-arrival times.

The most common way to model the job arrival process is as a *Poisson process* where the time between successive arrivals is exponentially distributed with parameter λ and independent of the past.

Let $U(i)$ the random process that characterizes the inter-arrival times between two jobs.

Let $\mathbb{P}(U_i > u)$ the probability that inter-arrival time is longer than u:

$$\mathbb{P}(U_i > u) = \exp(-\lambda u). \tag{2.9}$$

The average time $\mathbb{E}[U]$ between two successive arrivals is:

$$\mathbb{E}[U_i] = \frac{1}{\lambda}. \tag{2.10}$$

Service duration

The time it takes to serve a job is modeled by a stochastic process, called the *service process*. It can be described in two ways:

- the average number of jobs served per unit time noted as μ,
- the time it takes to serve a job. The average service duration is $1/\mu$.

An exponential distribution is usually used to model the service duration. Let S the random process that characterizes the service duration of a job, then the probability $\mathbb{P}(S > s)$ is given by the following:

$$\mathbb{P}(S \geq s) = exp(-\mu s) \tag{2.11}$$

By the arrival rate and service rate we can then define the server utilization using $\rho = \lambda/\mu$.

Servers and Waiting Buffer

Servers

Queuing systems can have a single or multiple servers that can serve jobs simultaneously. In the general case, in multi-servers systems the distribution of

service time is not uniform and each server can have different service distributions. However, it is common to assume that servers are all homogeneous and thus characterized by same service distributions.

Waiting buffers

Queues can have a *finite* or an *infinite* buffer. When the buffer is finite once it becomes full, the jobs may be blocked or as well dropped. In queuing theory terminology these are called *blocking systems*. When the buffer is infinite, the jobs are never blocked and these systems are called *pure waiting systems*.

Service disciplines for queues

The *service discipline* describes the order in which jobs are taken from the queue and selected for service. These are the following:

- Service disciplines without priorities
  - FIFO (First-In-First-Out)
  - LIFO (Last-In-First-Out)
  - RS (Random Service)
- Service disciplines that are *fair*: all jobs are served together and the service capacity they received is shared equally among them. This is called *Processor Sharing* or PS.
- Priority Service disciplines : there are jobs with higher priority than others.

One common way to present all different configurations is throughout *Kendall* notation.

Kendall Notation

The general form of the Kendall notation is $A/B/C/K/Z$ where

- A is inter-arrival time distribution
- B service time distribution
- C number of servers
- K is the system capacity

- Z is the service discipline

A and B can be

- M: Memory-less or Markovian (Exponential distribution)
- D: Deterministic (Constant distribution)
- G: General distribution (Arbitrary distribution)

The $M/M/1$ queuing model is the simplest case and it is introduced in the following section.

2.5.1 M/M/1

The most basic queuing model that is used is one with Poisson arrival distribution with rate λ, exponential service times with parameter μ, one server, infinite capacity and a FIFO service discipline. This is expressed under the Kendall notation as $M/M/1$ and it is represented in Figure 2.3.

Figure 2.3: M/M/1 queuing model

The full notation is $M/M/1/\infty/FIFO$ where default values are omitted.

In this one-server queuing model, it is proven that the average time $\mathbb{E}[T]$ between the arrival of the request and the response (mean response time) depends on λ and μ following the formula below:

$$\mathbb{E}[T] = \frac{1}{\mu - \lambda} \tag{2.12}$$

In practice data centers run on multiple servers and are connected through multiple network links. Furthermore, in a datacenter instead of a *First Come*

First Served (FCFS) service discipline there is *Processor Sharing* (PS) discipline with priorities. In complex systems such as the Internet, traffic does not follow a Poisson distribution but it is has been shown to follow a heavy-tailed distribution [Willinger 1998], [Pióro 2004]. Finally, service times need not be exponentially distributed. In that case, the analysis for the computation of processing time is part of the formula and the expressions for the distribution of the response time are not easy to obtain.

However, it is common to use $M/M/1$ queues as an approximation of complex systems because they are simple to reason with.

In a single server system when jobs arrive according to a Poisson process and job sizes can follow any distribution is that the mean response time is independent of service discipline if the discipline is one of FCFS, LFCS or Random [Harchol-Balter 2013].

Queuing theory allows to model more realistic architectures such as *server farms* that consist of a number of servers that concurrently handle incoming jobs.

The advantages of this architecture are firstly the *economies of scale* since it is cheaper to buy commodity hardware than acquire one very powerful server. Another benefit is that servers can be added or removed at any time if the capacity requirements change.

A server farm can have a *single central queue* of requests or it can be a system where there are *queues at the individual servers*. With a central queue, the next job is served directly when the server is free. In the second case, the next job is dispatched to one of the servers according to the *task assignment policy.*. The choice of the task assignment policy is important because it influences the response time.

The following are some of the task assignment policies that can be used in order to dispatch jobs.

- **Random** : each job is dispatched randomly to hosts,
- **Round-Robin** : each job goes to a a server in turn,
- **Join-the-Shortest-Queue** (JSQ) : a job is dispatched to the server with the fewest number of jobs. It is state-dependent since the number of jobs in each of the servers has to known to the dispatcher,
- **Site-Interval-Task-Assignment** (SITE) : each job is given a definition of "short", "medium" and "long" and"short" jobs are dispatched to the first host, "medium" to the second and so on,

- **Least-Work-Left** (LWL): a job is dispatched to the server with the least total remaining work

Central queue models $M/M/k$ is the shorthand Kendall notation of $M/M/k/\infty$ that models a server farm having a single queue with Poisson arrivals and exponential service times and an unbounded queue. This is often represented like in figure 2.4. In contrast, $M/M/k/k$ models a server farm with a bounded queue of capacity k where the $k+1$'th arrival is dropped by the system. An $M/M/\infty$ queue models a theoretical architecture where the server farm has an infinite number of servers. There is thus always a free server to process arriving jobs.

Individual server queues Another possible model is not to place the server farm behind a central queue but instead have each server used a dedicated queue (Figure 2.5). The jobs are assigned to a specific queue according to the task assignment policy, and remain in the queue the corresponding server processes them. This is known as the *frequency-division multiplexing*-(FDM) whereas the server $M/M/k$ model is called the *statistical-division multiplexing* [Harchol-Balter 2013].

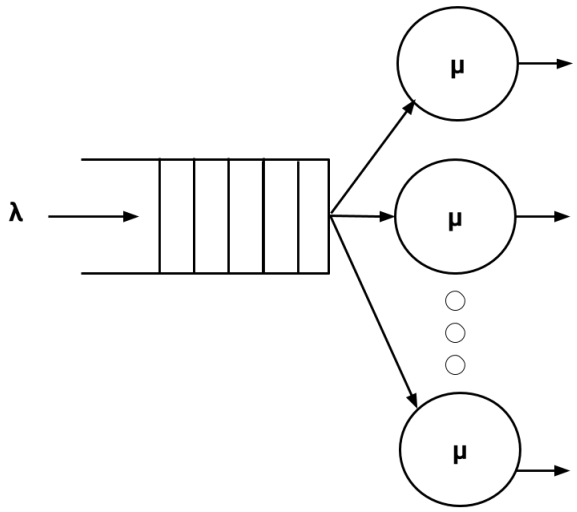


Figure 2.4: M/M/k queuing model

Performance comparison It is interesting to compare the performance characteristics of these different models and queuing theory provides analytic formulas that offer clear results beyond intuition.

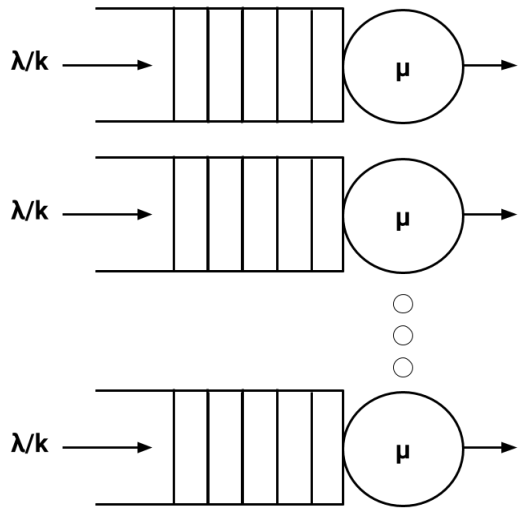


Figure 2.5: Frequency Division Multiplexing

Let an arrival rate λ and a total service rate $k\mu$ and load $\rho = \lambda/k\mu$. The mean response time under the FDM architecture is:

$$\mathbb{E}[T]^{FDM} = \frac{1}{\mu - \frac{\lambda}{k}} = \frac{k}{k\mu - \lambda} \tag{2.13}$$

Whereas the mean response time under the M/M/1 is:

$$\mathbb{E}[T]^{M/M/1} = \frac{1}{k\mu - \lambda} \tag{2.14}$$

This demonstrates that the $M/M/1$ is k times faster than FDM. However, there are still cases where one should use FDM to guarantee a specific service rate for some jitter-sensitive applications such as video or voice. In this case the job stream contains several jobs from the same users and merging them all into one central queue introduces a lot of variability in the response times [Harchol-Balter 2013].

Finally, it is also interesting to compare one $M/M/1$ queue of processing power $k\mu$ and $M/M/k$ queue. As we saw earlier, there are economies of scale going with the latter but we can still compare the performance.

Under light load, few jobs are dispatched and a server farm of k servers would have most of its servers idle. By comparison, one powerful server would be able to benefit from the light load to perform faster for these few jobs, thus making $M/M/1$ have a better response time than $M/M/k$. However under

heavy load the performance characteristics are similar to the computing resources of either the whole server farm of the single powerful server would be busy [Harchol-Balter 2013].

Chapter 3

Capacity Planning of Fog Computing Infrastructures

In this chapter we aim to optimize the cost of a newly designed Fog infrastructure. Our goal is to determine which Fog nodes among a proposed set should be installed in order to handle the expected traffic originating from base stations while keeping probabilistic response time guarantees.

In practice, the design of a Fog Computing infrastructure has to balance two conflicting factors. The first one is the importance of geographic diversity: the goal is to place micro data centres close enough to users to meet delay requirements. The other one is the size of the data centre: the goal here is to amortize the fixed costs of the site while benefiting from statistical multiplexing gains by serving the workload generated by the local population. Finding an optimal trade off between geographic diversity and data centre sizes is usually a challenging problem.

Despite the non-linearity of the delay constraints, we show that the problem can be formulated as a Mixed Integer Linear Programming problem. We first present a MILP formulation of the problem assuming that the infrastructure cost depends linearly on the capacities. We then extend this MILP formulation to arbitrary concave objective functions. Empirical results show that the optimal capacity-planning solution can be determined efficiently even for large-size problem instances, and that it can result in significant gains with respect to the solution in which user requests are always processed in the nearest data centre.

3.1 General context

There are several steps in the building of an infrastructure from the design phase to operations. We focus on the design phase. Examples of considerations in the design phase are for instance how much bandwidth to allocate in a given topology to the network. These belong to the category of topological design problems. Another example is how many replicas to assign to a given infrastructure. This is a family of problems known as *replica server placement*.

Building a large infrastructure like Fog Computing from scratch, deploying and operating it world-wide entails a colossal cost.

First, we have to consider the installation cost: the building of the Fog facilities themselves. Then comes the operating costs, which includes the staff. Furthermore, an important expenditure is the *energy* required to power and cool down the data centers.

Another cost item that needs critical consideration is the amount of compute capacity to assign. If a facility is assigned more than the required amount then the infrastructure is *overprovisioned*. Otherwise, if the facility is assigned less than the required amount, then the infrastructure is *underprovisioned* and it may not be able to serve all outstanding requests. *Capacity Provisioning* is an existing field of study that uses Queuing Theory and it addresses how much capacity to assign to servers.

An important aspect when deciding capacity is the *workload characteristics*. Key factors to consider are volume, time and location: the amount of traffic is changing throughout the day and can vary between geographical areas. For instance, commuting between home and work during the morning and evening hours generates a workload that needs to be considered in the design phase, as it may result in some resources being saturated.

As mentioned before, a key benefit of the Fog is bringing resources closer to the user to provide better latency. This allows to define strict *Service Level Agreements* (SLAs) in which applications are guaranteed to be served under a certain latency. This is especially important for latency critical applications. Without these SLAs, it would always be more efficient to opt for a central solution in the cloud by virtue of *economies of scale*.

These considerations fall under the umbrella of *Quality of Service* (QoS), and *Quality of Experience* (QoE). The former ensures a service is performed under a specific SLA, whereas the latter is about the user experience itself.

3.2 Related literature

In our study we construct a MILP model to represent the optimization problem of minimizing the infrastructure cost while optimising the routing strategy from base stations to Fog nodes.

As mentioned in the previous section, *topological design* happens in the initial phase of infrastructure design and focuses on defining the topology of the network composed of switches, routers, hubs, or locations for points-of-presence. The authors in [Pióro 2004] present a number of related problems.

There are as well analogous problems such as *Replica Server Placement* [A.S Tanenbaum 2006] and *Facility Location* problems that we have already presented in Chapter 2. In general these problems are NP-hard.

A number of studies have been devoted to the capacity planning of Fog infrastructures. In [Mondal 2017], the authors formulate a mixed integer non-linear programming for the placement and capacity planning of cloudlets, assuming as input a number of potential locations. The objective is to minimize cloudlet installation as well as networking cost. The authors model cloudlets as $M/M/1$ queues, which has the drawback that the processing time can become arbitrary low when the capacity is large. This assumes that the capacity of all the cloudlet servers can be pooled to serve a job. In practice, a job will be allocated a small fraction of the whole capacity and cannot use the capacity of the other servers even if they are idle. Further, this model will only give a lower bound on the capacity, which may not be a good approach for jobs requiring QoS guarantees. In addition, in contrast to the present work, the work in [Mondal 2017] considers only one class of jobs and does not take into account the temporal variations of traffic demands.

Another relevant work is [Kiani 2018]. Given the total capacity, the problem addressed by the authors amounts to distributing it among cloudlets and the cloud. Cloudlets are modeled as discrete-time fluid systems. In contrast to the present work, there is no routing decision from base stations to cloudlets. At each time instant, each cloudlet receives a random amount of traffic and, if the traffic exceeds the capacity of the cloudlet, the excess traffic is routed to the cloud. In this model, there is no infrastructure cost, and the goal is to optimize the quality of service of network flows. The authors discuss variations of the objective function based on the delay or loss probability.

Most of the works on the optimal design of Fog Computing infrastructures focus on the optimal placement of cloudlets and on traffic offloading to the cloud [Ceselli 2015] [Sun 2017] [Fan 2017][Mehta 2016] [Gelenbe 2012] [Xu 2016] [Jia 2017] [Xiao 2017b]. For instance, the authors in [Xiao 2017b] investigate how

to optimally select K mobile access points in which to install a cloudlet, assuming that each cloudlet has a fixed capacity and that a fraction of the traffic is offloaded to the cloud. As another example, [Sun 2017] studies the optimal offloading strategy of mobile devices to fixed-capacity cloudlets with the objective of minimizing the latency.

Some other works have addressed the capacity planning problem, but without any queuing model. For instance, the authors in [Xiao 2017b] suggest to use profiling and benchmarking tools to determine the resource requirements of applications from their workload. They assume that latency-sensitive applications are always executed in cloudlets, whereas other applications are executed in the cloud. In order to minimize the capacity required in each cloudlet, they solve a Knapsack problem.

3.3 Distributing versus Sharing

In the problem we study we consider the cost of the compute capacity. The decision of how much compute capacity to allocate in our model depends on three factors:

- the amount of traffic
- statistics of daily usage patterns
- the application's QoS constraints

Consider the simple scenario depicted in Fig. 3.1, in which two different base stations can route their traffic to two different micro data centers. The minimum latency is achieved with a distributed solution, in which the traffic of each base station is routed to the closest micro data center, so that base station B_1 (resp. B_2) routes all its traffic to data center D_1 (resp. D_2).

Now, assume that the daily pattern of the offered traffic at each base station is as shown in Fig. 3.2. In the distributed solution described above, the minimum amount of capacity to be provisioned at data centre D_1 (resp. D_2) corresponds to the peak hour of traffic for base station B_1 (resp. B_2), so that the total capacity to be provisioned is for $240 + 240 = 480$ jobs/s. In contrast, in the centralized solution where only one data centre is used, the total capacity to be provisioned should be only for 282 jobs/s. Thus the ratio of total capacities between the centralized and the distributed solutions is roughly $\frac{1}{2}$. In turns, this translates into an even lower ratio in terms of costs in favor of the centralized solution due to economies of scale which impacts not only capacity costs, but also

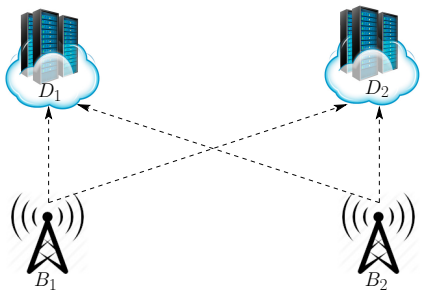


Figure 3.1: A simple scenario in which two base stations can route their traffic to two different micro data centres.

operating and energy costs. The issue is that, for some services with stringent delay requirements, a centralized solution might not be feasible.

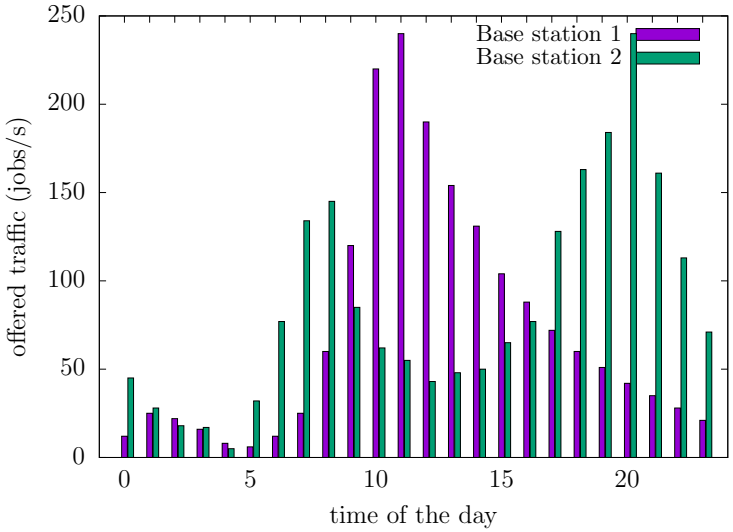


Figure 3.2: Daily pattern of the offered traffic by the two base stations, as a function of the hour of the day.

The ratio of compute capacities between a centralised and a distributed solution has been well studied with the first appearance of distributed systems, trying to solve how to efficiently organize compute power on processors among multiple machines. [A.S Tanenbaum 2006].

3.4 Motivation

This study explores the trade-off between the geographic diversity of incoming jobs and the amount of compute capacity to allocate to Fog nodes. Using our model Fog providers can build a minimum cost infrastructure and provide users an adequate *Quality of Experience*.

We aim to answer to the following questions:

1. What amount of compute capacity to install on each Fog node?

2. How to route traffic originating from the base stations?

We consider compute capacities for *classes of services* which can be provisioned for each service separately. In our formulation capacities are provisioned once for all the times in the day. However, the traffic can vary over time and base station can change their routing from one time slot to the other.

As input to the problem we have the following:

- Workload characteristics: in a design problem, a Fog provider can obtain in advance the statistics of traffic patterns, for example which zone is busier and the time of the rush hours.

- Network latency: we as well assume that *network latencies* to send traffic from a base station to each of the Fog nodes are known and fixed during the design problem. However, this is an approximation we make since network latencies are dynamic and depend on the network load.

We add probabilistic *Quality of Service* constraints making it a *Chance-Constraint Problem*: a category of optimization problems solved under uncertainty. Furthermore, we are assuming Poisson arrivals for the jobs, exponential service time distributions and we model Fog nodes as n parallel *M/M/1* queues. The problem is then solved as *Mixed Integer Linear Program*.

3.5 Problem Statement

We are given as input a set $\mathcal{D}$ of potential sites for installing micro data centres, as well as a set $\mathcal{B}$ of base stations.

The infrastructure supports a set $\mathcal{S}$ of S job classes. Let $\lambda_i^{k,t}$ be the class-k traffic originating from base station $i \in \mathcal{B}$ at time $t = 1, 2, \ldots, \tau$.

Routing Strategy

We define $x_{ij}^{k,t}$ as the amount of class-k traffic sent by base station i to micro data centre j at time t. These variables, which define the routing strategy at time t, have to satisfy the following constraints (3.1) and (3.2)

$$\sum_{j \in \mathcal{D}} x_{ij}^{k,t} = \lambda_i^{k,t}, \tag{3.1}$$

$$x_{ij}^{k,t} \geq 0. \tag{3.2}$$

These constraints can be interpreted as:

- Constraint (3.1) corresponds to the conservation of flow. Traffic of each service class, at each base station, at each time step should be distributed among all resulting allocations. No traffic is unaccounted for.
- Constraint (3.2) that traffic from base station i can be routed or not to Fog node j

We also define the binary variable $a_{i,j}^{k,t}$, which indicates whether base station i sends class-k jobs to data centre j at time t, and the binary variable $u_j^{k,t}$, which indicates whether class-k jobs are routed to data centre j, by imposing the following constraints on the feasible values of these variables.

$$\sum_i a_{i,j}^{k,t} \leq |\mathcal{B}| u_j^{k,t}, \tag{3.3}$$

$$u_j^{k,t} \leq \sum_i a_{i,j}^{k,t}, \tag{3.4}$$

$$x_{i,j}^{k,t} \leq \lambda_i^{k,t} a_{i,j}^{k,t}, \tag{3.5}$$

$$u_j^{k,t} \in \{0,1\}, \tag{3.6}$$

$$a_{i,j}^{k,t} \in \{0,1\}. \tag{3.7}$$

- Constraints (3.3) and (3.4) express that a base station will route traffic to a Fog node only if it is connected.
- Constraint (3.5): The variable $a_{i,j}^{k,t}$ is equal to 1 if at least one BS i sends class-k traffic to site j at time t.

Finally, the binary variable u_j defined by the following constraints

$$u_j^{k,t} \leq u_j, \quad \forall k, t, \tag{3.8}$$

will be used to determine whether data centre j has to be opened. Note that there is no need to enforce that $u_j = 0$ if $u_j^{k,t} = 0$ for all k and t because the objective functions that we consider are non-decreasing in u_j. In the following, the variables $x_{i,j}^{k,t}$, $a_{i,j}^{k,t}$, $u_j^{k,t}$ and u_j shall be referred to as the *routing variables* of the problem.

A feasible routing strategy is a set of values for these variables satisfying (3.1)-(3.8).

3.5.1 Performance Requirements

The performance requirements are related to the quality of service of jobs processed by the servers of the FC infrastructure. The system designer aims at determining the capacities of the data centres in such a way that most class-k jobs be served in a maximum acceptable processing time T_k. More precisely, let

- $S_j^{k,t}$ be the processing time of class-k jobs at data centre j and at time $t = 1, 2, \ldots, \tau$, and
- let $\ell_{i,j}^k$ be the network time, that is, the time it takes to send a job request from base station i to data centre j plus the time it takes to receive the reply.

As mentioned before, for simplicity, we assume that $\ell_{i,j}^k$ is a fixed communication delay which does not depend on the network load. In contrast, the processing time $S_j^{k,t}$ is a random value whose distribution may depend on the load of the data centre and on the class of the job. The term $S_j^{k,t} + \ell_{i,j}^k$ then represents the total time it takes for a class-k request sent by node i at time t to be received and processed by micro data centre j, plus the time it takes to receive the reply.

The goal is to design the system in such a way that the probability that this time be strictly greater than T_k be lower than a given value δ_k, that is, in such a way that

$$\mathbb{P}\left(S_j^{k,t} + \ell_{i,j}^k \geq T_k\right) \leq \delta_k,$$

for all base stations i sending class-k jobs to micro data centre j at time t. In other words, the above delay constraints should be enforced only for those i such

that $a_{i,j}^{k,t} = 1$ and for those values of j, k and t such that $u_j^{k,t} = 1$. This can be done by imposing that

$$\mathbb{P}\left(S_j^{k,t} \geq T_k - \max_i \ell_{i,j}^k a_{i,j}^{k,t}\right) \leq \delta_k + 1 - u_j^{k,t}. \tag{3.9}$$

Note that if no class-k jobs are routed to site j at time t (that is, $u_j^{k,t} = 0$), the above constraint is redundant. Otherwise, it imposes that $\mathbb{P}\left(S_j^{k,t} \geq T_k - \max_i \ell_{i,j}^k a_{i,j}^{k,t}\right) \leq \delta_k$, as expected.

We denote by c_j the number of compute servers installed in site $j \in \mathcal{D}$. It has to be big enough so that the latency constraints (3.9) are satisfied. We assume the following cost structure:

- An opening cost β_j is incurred if capacities are installed in data centre j, that is, if $u_j = 1$.
- The cost of installing c servers in data centre j is $g_j(c)$, where g is a given continuous function, which is often chosen concave to express economies of scale in favour of large data centres. Note that $g_j(c)$ includes the cost of purchasing the capacity c, but can also include energy and maintenance costs for operating it.

The problem can now be formally stated as follows

$$\begin{aligned} &\text{minimize} \sum_{j \in \mathcal{D}} \beta_j u_j \;+\; g_j(c_j) \\ &\text{subject to} \;\; \text{constraints (3.1)} - \text{(3.9)}. \end{aligned} \tag{CAPA}$$

where the precise form of the constraint (3.9) obviously depends on the queuing model which is assumed for Fog nodes.

Separate Resource Provisioning Per Service

As we have already mentioned in Section 3.4 we consider the case where the system designer provisions resources for each service separately.

We let c_j^k be the number of compute servers provisioned for handling class-k and $\frac{1}{\mu_k}$ be the mean processing time of a class-k job on one of these servers[1]

[1] In the following, we consider c_j^k as a continuous parameter. In practice, the value that should be used is $\lceil c_j^k \rceil$.

Obviously, we have

$$c_j = \sum_k c_j^k \tag{3.10}$$

We can analyse separately the optimal capacity to be provisioned for each class. As a consequence, in sections 3.5.2 and 3.5.3 below we consider only one class of jobs, and we drop the index k.

3.5.2 Queuing Model

In Chapter 2 we have introduced some basic queuing models. The simplest one is called $M/M/1$ and uses a Poisson arrival rate and exponential service time. We have as well introduced different task allocation strategies such as Join-the-Shortest-Queue (JSQ) where the number of servers is known to the dispatcher in contrast with Bernoulli routing where jobs are routed to server j with a probability that does not depend upon the number of active tasks in each server. In our model we have the following:

Arrival Process

We assume that job requests arrive according to a Poisson process and that an incoming job is routed with probability $1/c_j$ to any of the servers using what is known as Bernoulli routing. The rate of the arrival process is defined as $y_j^t = \sum_{i \in \mathcal{B}} x_{i,j}^t$ is the rate at which job requests arrive at data center j at time $t = 1, \ldots, \tau$

Service Process

The service time of a job on a server will be assumed to be exponentially distributed with mean $1/\mu$.

It follows from these assumptions that the servers provisioned at data-centre j for the considered class are modelled as c_j parallel $M/M/1/\infty$ queues [Kleinrock 1975], and therefore that

$$\mathbb{P}\left(S_j^t \geq z\right) = e^{-(\mu - y_j^t/c_j)\, z} \tag{3.11}$$

3.5.3 Optimal capacity for a fixed routing strategy

Our first step is to analyze the capacity required at a given data center, say j, for a fixed routing strategy satisfying (3.1)-(3.8). We first note that, for stability

reasons, we should have $y_j^t/c_j < \mu$, that is, $c_j > y_j^t/\mu$. This condition is however not sufficient, as formally stated below.

Lemma 3.5.1 *There exists $c_j > y_j^t/\mu$ satisfying the latency constraint of jobs if and only if the routing strategy is such that*

$$l_{i,j}a_{i,j} < T - \frac{\log(\frac{1}{\delta})}{\mu}, \quad i \in \mathcal{B}, t = 1, \ldots, \tau \tag{3.12}$$

Proof 3.5.2 *With (3.9) and (3.11), we obtain*

$$\frac{y_j^t}{c_j} \leq \mu - \frac{\kappa}{T - \ell_{i,j}a_{i,j}^t}, \tag{3.13}$$

where $\kappa = \log(\frac{1}{\delta})$. Inequality (3.13) ha a non-negative solution c_j if and only if the RHS is non-negative, that is, if and only if $l_{i,j}a_{i,j} < T - \frac{\kappa}{\mu}$

The condition in Lemma 3.5.1 merely imposes that $a_{i,j}^t = 0$ whenever $\ell_{i,j} \geq T$. Provided that this condition is met, Lemma 3.5.3 below gives the optimal number of servers to install in data center j for known values of the other variables.

Lemma 3.5.3 *Given the values of the routing variables, the minimum number of servers required to satisfy the latency constraint of jobs is*

$$c_j = \max_{t,i} \left\{ \frac{y_j^t}{\mu - d_{i,j}} a_{i,j}^t \right\}, \tag{3.14}$$

where $d_{i,j} = \log(\frac{1}{\delta})/\left[T - \ell_{i,j}\right]$.

Proof 3.5.4 *Inequality (3.13) can equivalently be written as follows*

$$c_j \geq \frac{y_j^t}{\mu - \kappa u_j^t / \left[T - \max_i \left(\ell_{i,j}a_{i,j}^t\right)\right]}. \tag{3.15}$$

Indeed, if $u_j^t = 1$, the RHS of (3.13) and (3.15) are obviously equal. If on the contrary $u_j^t = 0$, then it follows from (3.3) and (3.5) that $x_{i,j}^t = 0$ for all i and therefore that $y_j^t = 0$, which implies that the equality between the RHS of (3.13) and (3.15) holds.

We shall now make use of the following result.

Lemma 3.5.5 *For any values of the routing variables satisfying (3.1)-(3.7), it holds that*

$$\frac{\kappa}{T - \max_i \left(\ell_{i,j} a_{i,j}^t\right)} u_j^t = \max_i \left(\frac{\kappa}{T - \ell_{i,j}} a_{i,j}^t\right) \tag{3.16}$$

Proof 3.5.6 *Noting that the function $x \to \frac{\kappa}{T-x}$ is strictly increasing over $[0, T)$, we obtain*

$$\frac{\kappa}{T - \max_i \left(\ell_{i,j} a_{i,j}^T\right)} u_j^t = \max_i \left(\frac{\kappa}{T - \ell_{i,j} a_{i,j}^t}\right) u_j^t.$$

If $u_j^t = 0$, then from constraint (3.3) we have that $a_{i,j}^t = 0$ for all i, and hence equality (3.16) is satisfied. If on the contrary $u_j^t = 1$, then constraint (3.4) implies that there exists k such that $a_{k,j}^t = 1$, and hence that

$$\max_i \left(\frac{\kappa}{T - \ell_{i,j} a_{i,j}^t}\right) \geq \frac{\kappa}{T - \ell_{k,j}} > \frac{\kappa}{T}.$$

Since $\frac{\kappa}{T-\ell_{i,j} a_{i,j}^t} = \frac{\kappa}{T-\ell_{i,j}} a_{i,j}^t$ when $a_{i,j}^t = 1$, and $\frac{\kappa}{T-\ell_{i,j} a_{i,j}^t} = \frac{\kappa}{T}$ when $a_{i,j}^t = 0$, we conclude that equality (3.16) is also satisfied when $u_j^t = 1$.

From Lemma 3.5.5, it follows that

$$\begin{aligned} c_j &\geq \frac{y_j^t}{\mu - \max_i \left(d_{i,j} a_{i,j}^t\right)}, && (3.17)\\ &= \max_i \left\{\frac{y_j^t}{\mu - d_{i,j} a_{i,j}^t}\right\}, && (3.18)\\ &= \max_i \left\{\frac{y_j^{k,t}}{\mu_k - d_{i,j}^k} a_{i,j}^{k,t}\right\}, && (3.19) \end{aligned}$$

where the equality between (3.17) and (3.18) follows from the fact that the function $z \to \frac{y_j^t}{\mu - z}$ is strictly increasing over $[0, \mu)$. The last equality is proved by considering two different cases:

- *If $a_{i,j}^t = 0$ for all i, then it follows from (3.5) that $x_{i,j}^t = 0$ for all i and therefore that $y_j^t = 0$, which implies that the equality between (3.18) and (3.19) holds.*

- *if* $a^t_{i,j} = 1$ *for some* i*, then the maximum value in* (3.18) *is obtained for some* k *such that* $a^t_{k,j} = 1$*, and this value is equal to the value obtained in* (3.19)*, which proves that the equality is also valid in this case.*

Finally, we conclude the proof by observing that the inequality (3.18) *has to be satisfied for all values of* t*, which yields* (3.14)*.*

It follows from Lemma 3.5.3 that the optimal capacity at data center j is the minimum value satisfying the following linear inequalities

$$c_j \geq \frac{y^t_j}{\mu - d_{i,j}} - M\left(1 - a^t_{i,j}\right), \tag{3.20}$$

$$c_j \geq 0, \tag{3.21}$$

where M is any constant sufficiently large for the RHS of (3.20) to be negative whenever $a^t_{i,j} = 0$.

3.5.4 Linear objective function

In this section, we consider the case where $g_j(c) = \alpha_j\, c$, for some constant α_j. It directly follows from Lemma 3.5.3 that the optimal solution of problem (CAPA) is obtained by solving the following Mixed Integer Linear Programming (MILP) problem

$$\text{minimize} \sum_{j \in \mathcal{D}} (\beta_j\, u_j \;+\; \alpha_j\, c_j) \tag{CAPA-PL}$$

$$\text{subject to} \;\; \text{constraints } (3.1) - (3.8), (3.10), (3.12), (3.20) - (3.21).$$

3.5.5 Piecewise linear objective function

As mentioned in the introduction, the function $g_j(c)$ is often a non-linear concave function to express economies of scale in favour of large data centres. The minimization of a non-linear concave objective function is in general a challenging problem, in particular when integer variables are involved. However, with the increasing efficiency of MILP software tools, an interesting alternative is to use a piecewise linear (PWL) approximation of the original non-linear function.

The PWL approximation of a function $f(x)$ over an interval $[x_{min}, x_{max}]$ is obtained by introducing a number n of sampling coordinates $x_1, \ldots, x_n$ such that

$x_1 = x_{min}$ and $x_n = x_{max}$. The function $f(x)$ is then approximated by the collection of linear segments $[(x_i, f(x_i)), (x_{i+1}, f(x_{i+1}))]$. Figure 3.3 illustrates the quality of the approximation obtained for two concave functions, $f_1(x) = \log(1+x)$ and $f_2(x) = \frac{3}{2} + \frac{1}{4}\sqrt{x}$, over the interval $[0, 50]$.

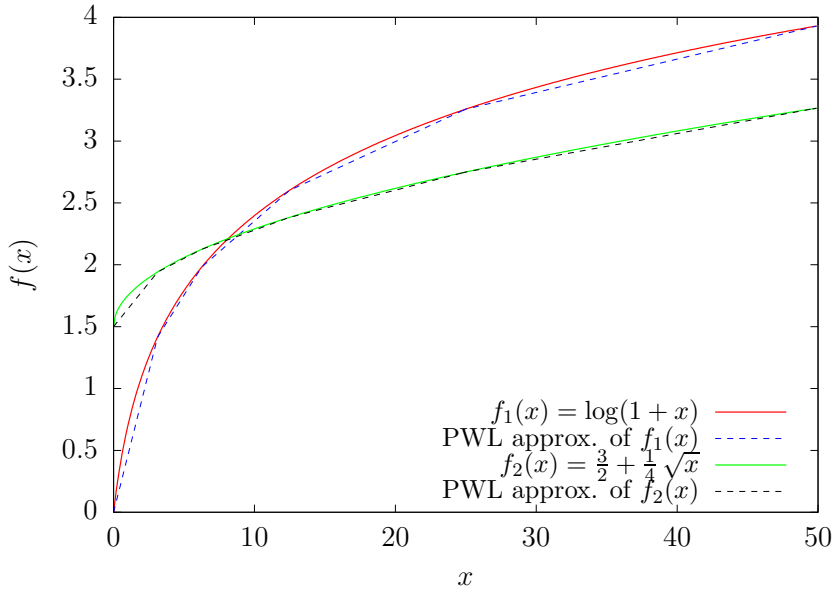


Figure 3.3: PWL approximations of the functions $f_1(x) = \log(1+x)$ and $f_2(x) = \frac{3}{2} + \frac{1}{4}\sqrt{x}$ over the interval $[0, 50]$. The number of sampling coordinates is $n = 5$, and they have been generated as follows: $x_1 = 0$ and $x_i = 2^{-(n-i)}50$ for $i = 2, \ldots, n$.

If n is the number of linear segments of the approximation, the above technique can be applied to our problem by introducing n continuous variables and $n-1$ binary variables, as described in [D'Ambrosio 2010]. We note however that most modern MILP solvers are capable of directly handling PWL objective functions, usually using the concept of Special Ordered Set.

3.6 Experimental Results

We now describe the results that were obtained with the proposed algorithms. We first consider a very simple scenario in Section 3.6.1. The numerical results obtained with a larger number of base stations are presented in Section 3.6.2. But

before we present the TCP transfer times of a file that explains our methodology to compute communication times in Table 3.2

TCP transfer times

In this section, we study the transfer time of a file with TCP as a function of the RTT and the file size. To this end, we use a simple deterministic model which was proposed in [Aarag 2001]. The model assumes an ideal environment in which no losses occur and round-trip times are constant. The model can thus be used to estimate lower bounds on the transfer time in most standard TCP implementations.

Let N be the total number of packets transmitted, T be the total transfer time (ms) and RTT be the round-trip time between the sender and the receiver (ms). Let also W_s be the TCP slow-start threshold and W_{max} be the maximum window size, both being measured in packets. Using simple arguments, it is then possible to show that the number of packets transmitted in the TCP slow-start phase is

$$N_{ss} = 2\,W_s - 1,$$

whereas the number of packets transmitted in the congestion-avoidance phase is

$$N_{ca} = (W_{max} - W_s - 1)\ \frac{W_{max} + W_s}{2}.$$

It follows that for short files containing $N \leq N_{ss}$ packets, the transfer time is given by

$$T = (\lceil \log_2(N) \rceil + 1)\ RTT,$$

where $\lceil x \rceil$ denotes the smallest integer greater than x. Files with more than N_{ss} packets but with less than $N_{ss} + N_{ca}$ packets are completely transmitted before the end of the congestion avoidance phase. In this case, the transfer time is given by

$$\begin{aligned} T \;=\; & \{\log_2(W_s) + 1 \\ & + \left\lceil \left(\sqrt{(2W_s+1)^2 + 8(N - 2W_s + 1)} \right. \right. \\ & \left. \left. -(2W_s+1)) \,/2 \right\rceil \right\}\ RTT. \end{aligned}$$

Finally, for files of size strictly greater than $N_{ss}+N_{ca}$ packets, $N-(N_{ss}+N_{ca})$ packets are transmitted in the steady-state phase. In this phase W_{max} packets

are transmitted per RTT. A slightly more complex formula can be derived to compute the transfer time of such files (see equation (4) in [Aarag 2001]).

Note that, independently of the file size, the transfer time is linear in RTT. Fig. 3.4 shows how the transfer time evolves as a function of the file size for two different values of the RTT, 10 ms and 20 ms. It was assumed that W_s = 16 packets and W_{max} = 64 packets, and that the packet size is 1500 Bytes. Interestingly, we note that when the RTT is 20 ms, the transfer time of only 3 packets (4.5 kB) is already 60 ms. When the RTT is 10 ms, a file of size 13.5 kB (9 packets) is transferred in 50 ms.

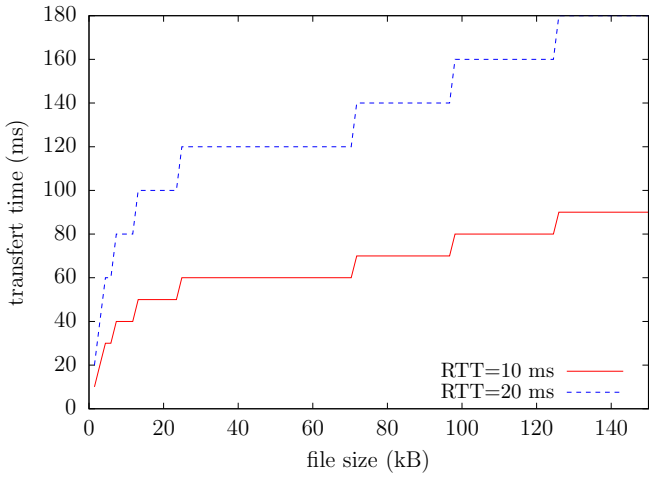


Figure 3.4: Transfert time with TCP as a function of the file size.

3.6.1 Simple scenario

We first consider a simple scenario with three data centres and two base stations, which are located as shown in Fig. 3.5. The first two data centres, located in Aulnay-sous-Bois and Corbeil-Essones, in France, are potential data centres. The cost for opening them are $\beta_1 = \beta_2 = 100$, and the cost of one unit of capacity is $\alpha_1 = \alpha_2 = 1$. In contrast, the data centre in London is an existing large public data centre. Therefore, there is no opening cost associated to this data centre ($\beta_3 = 0$) and we assume that, due to economies of scale, the cost of an individual compute server is only $\alpha_3 = \frac{3}{4}$. Note that each base station is in immediate vicinity of a data centre. The distance from the base station 1 in Rosny-sous-Bois to the data centre in Aulnay-sous-Bois (resp. Corbeil-Essonnes) is 8.9 Km

Table 3.1: Characteristics of job classes. Times are given in seconds, and n_k represents the number of packets send by a class-k request.

	$1/\mu_k$	T_k	δ_k	n_k
Class 1	0.01	–	–	2
Class 2.	0.1	2.0	0.1	6

(resp. 32.6 Km), and the distance from the base station 2 in Evry to the data centre in Corbeil-Essonnes (resp. Aulnay-sous-Bois) is 4 Km (resp. 35.7 Km).

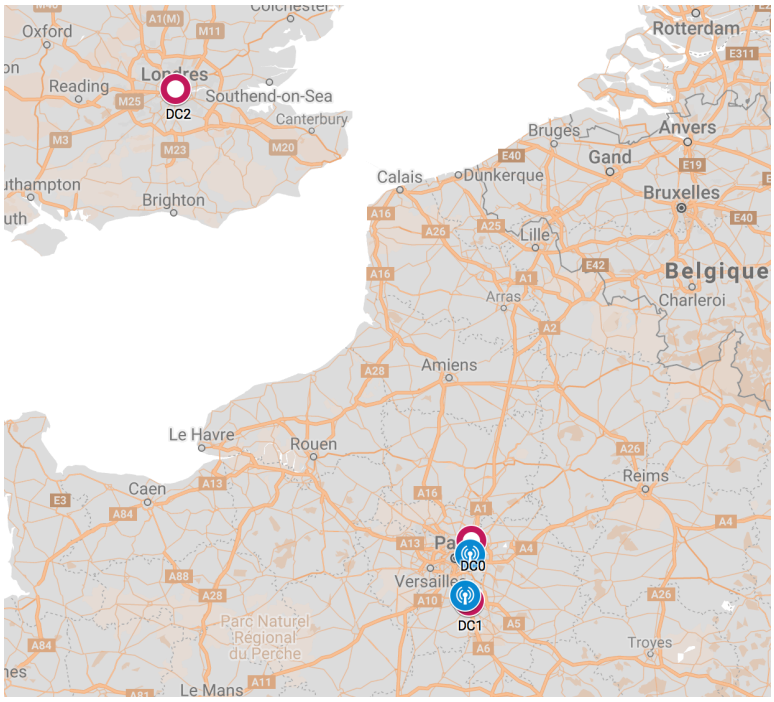


Figure 3.5: Locations of data centers and base stations.

The workload is composed of two classes of jobs. The first class of jobs corresponds to real-time jobs, whereas the other ones are best-effort jobs with far less stringent requirements. The parameter values used in our experiments are given in Table 3.1. Note that the values of the maximum end-to-end latency T_1 and the threshold probability δ_1 for the real-time class are not given in Table 3.1, because we will vary their values in the following.

We also assume that the communication latency between two points at a

Table 3.2: Communication times (ms) between base stations and data centres.

	Aulnay	Corbeil	London
Rosny	33 / 43	39 / 52	134 / 178
Evry.	41 / 54	31 / 41	140 / 187

distance of d kilometres from each other is $10 + 0.1 \times d$ ms, which, according to the idealized deterministic model in [Aarag 2001], yields the TCP transfer times given in Table 3.2, where in each cell the first (resp. second) value is the communication time for the first (resp. second) class of jobs. Regarding real-time jobs, the offered traffic of each base station evolves as shown in Fig. 3.2, but with values which are scaled by a factor 10. or simplicity, we assume that the class-2 offered traffic of each base station is constant over time, and equals to $2,000$ jobs/s.

We first consider the case where the infrastructure cost is linear in the data centre capacities, that is,

$$100 \times (u_1 + u_2) + c_1 + c_2 + \frac{3}{4} \times c_3 \tag{3.22}$$

Our goal is to compare three different solutions:

- the first one is the optimal solution, which is obtained as the solution of the MILP problem (CAPA-PL),
- the second one is the fully distributed solution in which each base station is assigned to the nearest data centre. This solution is obtained by adding to problem (CAPA-PL), for each base station i, the constraints $a_{i,j} = 1$ if data centre j is the nearest one to base station i, and $a_{i,j} = 0$ otherwise.
- the third one is the minimum-cost centralized solution in which only one data centre is used. This solution is obtained by adding to problem (CAPA-PL) the following constraints:

$$\begin{aligned} \sum_t \sum_{(k',i') \neq (1,1)} a_{i',j}^{k',t} &= \tau \, [S|\mathcal{B}| - 1] \, a_{1,j}^{1,1}, \quad \forall j, \\ \sum_j a_{1,j}^{1,1} &= 1. \end{aligned}$$

Note that for low values of the maximum latency T, the problem might become infeasible with these additional constraints.

Using the MILP solver Gurobi [Gurobi Optimization 2018], we computed the cost of each of the above solution for $\delta_1 = 10^{-2}$ and $\delta_1 = 10^{-4}$, and for different values of the maximum latency T_1 between 69 ms and 300 ms. The results are reported in Fig. 3.6. As expected, the fully distributed solution is optimal for low values of T, whereas the minimum-cost centralized solution is either infeasible or very expensive. For instance, for $\delta = 0.01$ and $T = 80$ ms, the centralized solution is about 17% more expensive than the optimal one. However, as T increases, the centralized solution quickly becomes the optimal one, whereas the fully distributed one is significantly more expensive. For $\delta = 0.01$, the additional cost is +91% for $T = 300$ ms, but it is already +48% for $T = 100$ ms.

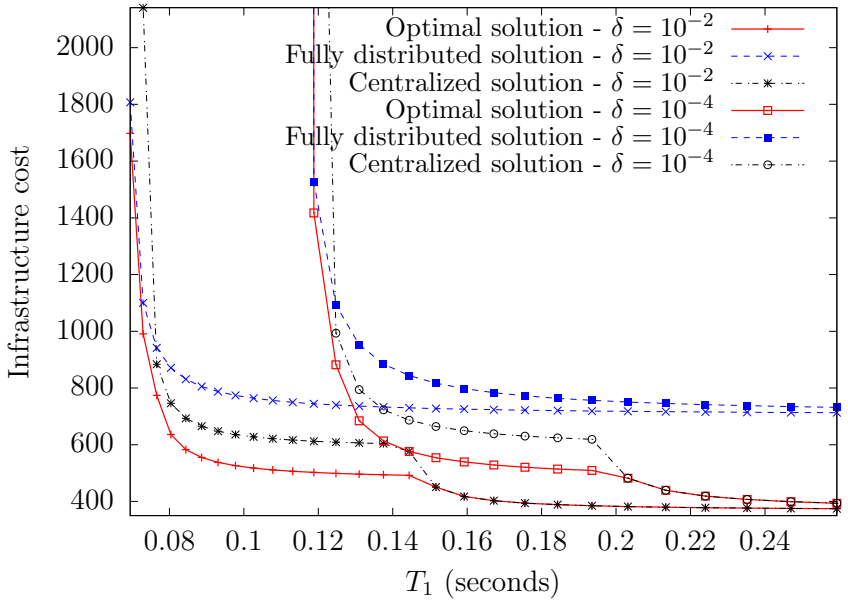


Figure 3.6: Cost of the Fog infrastructure as a function of the latency requirement of real-time jobs for the cost function given in (3.22).

Fig. 3.7 shows the probability that the end-to-end processing delay of jobs be greater than the allowed value as a function of the time of the day when $T = 100$ ms and $\delta = 0.01$. In this case, the optimal solution sends all real-time jobs to the data centre in Corbeil-Essonnes. Note that these probabilities fluctuate over time but never exceed δ.

Let us now evaluate the effect of economies of scale on the structure of the

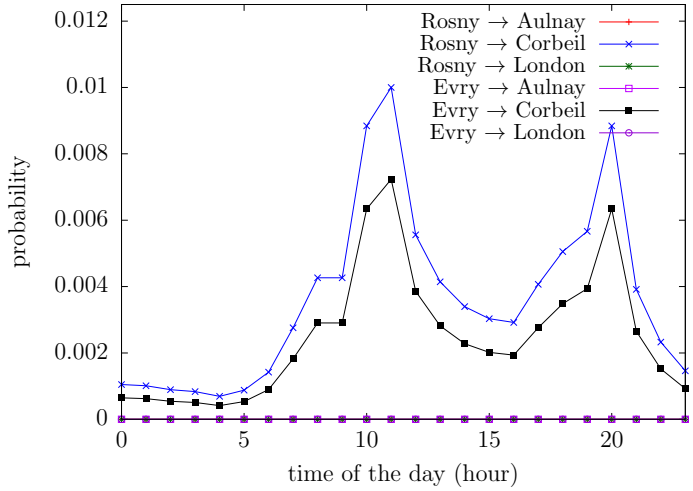


Figure 3.7: Probability that the end-to-end delay in the optimal solution be greater than the maximum allowed value as a function of time.

optimal solution. We now assume that the goal is to minimize

$$5 \times (u_1+u_2)+\log(1+c_1)+\log(1+c_2)+\frac{3}{4} \times \log(1+c_3). \tag{3.23}$$

The results obtained for this cost function are reported in Fig. 3.8. Note that the drop in the cost of the optimal solution at $T_1 = 190$ ms correspond to the point where the public cloud in London can host all the traffic. We remark that, as expected, the minimum-cost centralized solution is optimal whenever it is feasible. For $\delta = 0.01$ and $T = 80$ ms (resp. $T = 300$ ms), the cost of the fully distributed solution is 76% (resp. 323%) greater than that of the optimal one.

3.6.2 Larger number of base stations

We now build upon the previous scenario to design scenarios in which there is a larger number of base stations. We consider 29 base stations and 3 data centres, which are located as shown in Fig. 3.9. Note that for convenience, the data centre located in London is not shown in Fig. 3.9. Most of the base stations are in the area around Paris, but some of them are located a bit farther.

We consider the same classes of jobs than in the previous scenario (cf. Table 3.1) with $T_1 = 105$ ms and $\delta_1 = 0.01$. For values of T_1 below 105 ms, the delay requirement of real-time requests sent by the base station located in Tours cannot

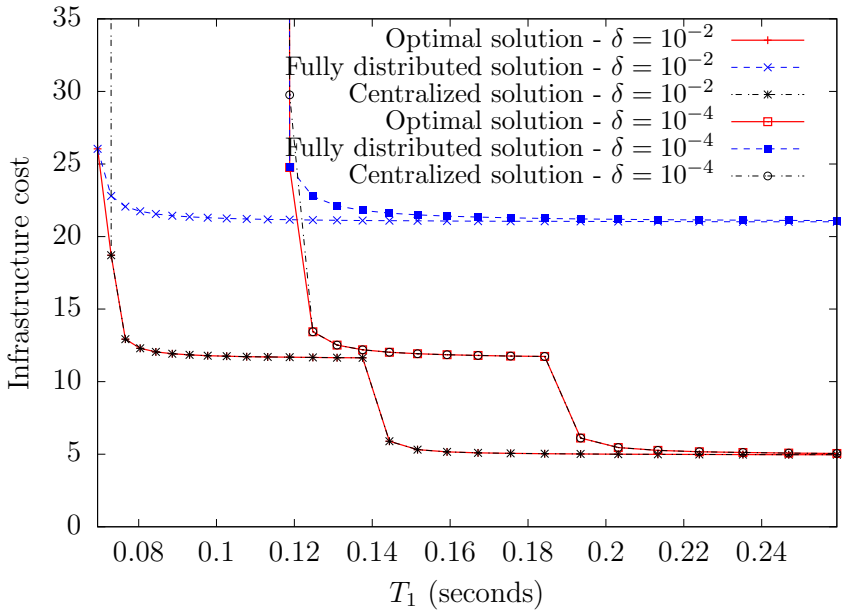


Figure 3.8: Cost of the Fog infrastructure as a function of the latency requirement of real-time jobs for the cost function given in (3.23).

be met. To generate random problem instances with realistic traffic patterns, we have used a spatio-temporal model inspired from [Wang 2015], in which a sinusoid superposition model is used to capture the temporal traffic variation, whereas a normal distribution is used for spatial traffic modelling at each epoch.

In a first scenario, we consider only the 5 first base stations, then in a second scenario we consider only the 10 first base stations, etc., until all 29 base stations are included in the sixth and last scenario. We have randomly generated 16 problem instances for each scenario. We have limited the total time expended by the solver gurobi to 5 minutes per problem instance. To avoid spending too much time in proving optimality, we also have set the relative gap of Gurobi to 2%, so that it terminates (with an optimal result) when the gap between the lower and upper objective bounds is less than 0.02 times the upper bound.

As before, we first consider the case when the infrastructure cost is linear in the capacities. Fig. 3.10 shows the minimum, maximum and average values of the relative gap in percent between the costs of the optimal solution and the other solutions as a function of the number of base stations. Surprisingly, we observe that the fully distributed and the centralized solutions are always around 30% more expensive than the optimal one. In most cases, the time limit of 5 mn was

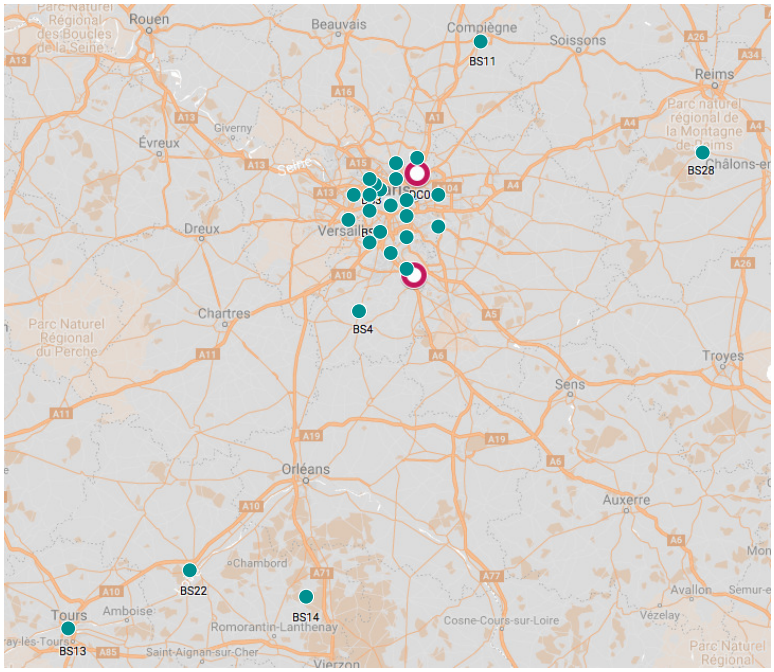


Figure 3.9: Locations of data centers and base stations for the third scenario.

reached by the solver. We however believe that the optimal solution was found in a few seconds in most cases by Gurobi, which was then not able to prove the optimality of the solution within the allocated timeframe.

The results obtained for a logarithmic cost function are reported in Fig. 3.11. The time limit of 300 seconds was always reached, which means that there is no optimality certificate for the solution obtained. As expected, the relative gap between the optimal cost and the cost of the centralized solution is less important than for a linear objective function. However, significant differences are still observed in Fig. 3.11 for more than 15 base stations.

3.7 Conclusion

We have shown that the optimal capacity-planning of micro data centres used in Fog Computing can be formulated as MILP problem, which can be solved efficiently even for large-size problem instances. Numerical results show that significant cost savings can be obtained with respect to the solution in which user requests are always processed in the nearest data centre, and with respect

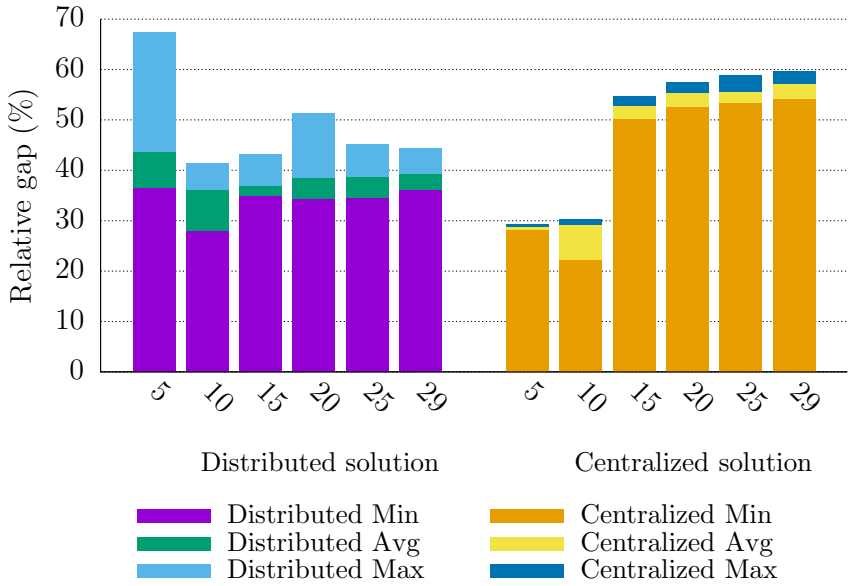


Figure 3.10: Relative gap in % between the costs of the optimal solution and the other solutions as a function of the number of base stations for a linear cost function.

to the minimum-cost centralized solution.

A possible extension is to consider situations in which the capacity of individual compute servers are shared among several classes of jobs, using for instance a strict priority mechanism or another more advanced resource sharing mechanism. One challenging problem is to dimension the system when the service times of jobs are not exponentially distributed. Since there are no known formulas for the distribution of the processing time in the general case, we intend to look at analytical approximations that can help dimension the system. Further, another consideration is to use more advanced load-balancing policies than Bernoulli routing for the distribution of jobs inside a data centre, such as for instance policies based on Power of Two Choices or Join the Shortest Queue.

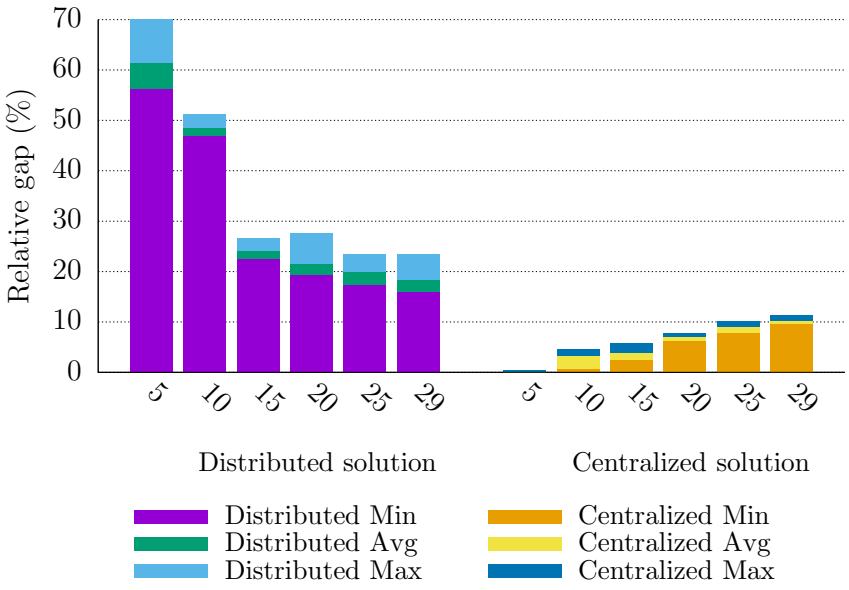


Figure 3.11: Relative gap in % between the costs of the optimal solution and the other solutions as a function of the number of base stations for the logarithmic cost function.

Chapter 4

Service Migration in Fog Networks

In this chapter, we present our contribution on the reduction of the infrastructure cost resulting from service migrations in the Fog.

One of the challenges that needs to be addressed is the maintenance of service continuity for mobile users. To ensure this continuity under a *Service Level Agreement* (SLA) with low latency constraints, the infrastructure providing the service must remain reasonably close to the user.

If the user moves in a way that the latency requirements can no longer be satisfied, we need to be able to serve the same service from a new location respecting the constraints. In order to do so, the active session should be transferred from one Fog server to another. We call this operation a *service migration*.

While frequent service migrations can improve the quality of service by keeping the Fog node close to the user, they introduce network and compute costs. Also, the overall cost of infrastructure increases with the number of Fog nodes and one would expect the number of migrations to increase with more Fog nodes. Therefore, there is benefit in ensuring the least amount of migrations occur while providing the adequate level of quality of service.

The model we present aims to minimize the cost of service migrations for a given set of base stations and Fog nodes they are connected to and handover statistics between the base stations. An extension of the model that reflects better reality is suggested at the end of the chapter.

While our work abstracts the underlying implementation of service migrations, in the following sections we first present some background on the subject and then present our contribution.

4.1 Service Migration

4.1.1 General context

Connected car applications are composed of the client-side application running on the car's embedded computer and a server-side component built as a distributed system hosted on the Fog infrastructure.

In the Cloud or the Fog, a *backend* service is made of a runnable and of the associated state. The *runnable* is usually packaged in a VM (*Virtual Machine*) or a *Linux Container*[1].

The *state* or *data object* is the data used by the runnable and it has a *persistent* component, which is usually stored in a database, and a *transient state*, which can not be recovered when the service is instantiated again. Nowadays most applications are designed as *stateless*. Services are called *stateless* when all the information required for the service to function properly is persisted in a database. Stateless services have a number of advantages such as allowing several instances of the service running concurrently, for instance to balance load or to provide fault tolerance. This is known as replication and is closely related to service migration, where instead of relocating a service instance, several instances are running concurrently [Flinn 2009].

A service migration entails the move of a running service from one server to another. In order to do so, the runnable and the associated state must be transferred. If the service is stateless the migration can be viewed as an instantiation of the runnable on a new server. If there is state persisted in a database, it should also migrate its data in case the SLA is not met.

Another characteristic of services to take into account is *tenancy*: services can either be designed for processes and storage to handle either one or many users. Each user is called a *tenant* and this kind of architecture is called a *multi-tenant architecture* since each service can run for several users or clients. This architecture helps reduce the infrastructure and operational maintenance overhead.

4.1.2 Study context

In our problem we are taking a user-centric approach, where we focus on data objects (user sessions and related service data) that need to be migrated between Fog nodes. We do not concern ourselves with the technical details of how data

[1]There are more modern alternatives such as WebAssembly modules. [was]

objects are implemented, or whether the service is implemented using virtual machines, containers etc.

In addition, it is not our concern whether this data object must be entirely migrated, or whether the service is running for one or many users. The fog computing infrastructure is composed of a number of base stations which are connected to a number of Fog nodes which in turn process the service requests. Fog nodes can be heterogeneous and have a different OS, computing capacity, number of servers, VMs, hardware etc. As presented so far, we do not consider base stations being Fog processing nodes, even though they may also provide services relevant to maintain the Fog running such as working with handover events to trigger service migrations. Base stations also use their networking data to decide which Fog node to route their traffic to or, more appropriately, decide where a data object should be migrated depending on its SLA.

The general picture in our model is that a car is connected using the cellular network to a number of base stations. Base stations maintain a list of candidates Fog nodes with allowed thresholds for SLA. Whenever a car moves to another base station, the data object may be migrated to another Fog node satisfying the SLA. An approach is to always keep migrating the service towards the fog node that is the closest to the current position. However, frequent service migrations have a migration and network cost. Bandwidth or capacity are some examples of metrics that express this cost but intuitively, the more migrations are triggered, the bigger this cost. In our work we look into ways to reduce this cost by studying how to minimize the number of the data object migrations triggered.

Instead of deciding of the appropriate Fog node to migrate to when a handover happens, the Fog controller computes the optimal set in advance when the itinerary is shared or predicted based on statistics.

4.2 Related literature

There are different types of service migration discussed in the literature, including the migration of virtual machines (VMs), containers or processes but in this section we focus mostly on mobility induced service migrations. An interesting study on virtual machine migrations with a section dedicated to mobility is [Zhang 2018a].

[Rejiba 2019] is a survey covering some of the state-of-the-art on mobility induced service migrations. The authors present recent advances where service migration has been studied on divers paradigms as cloudlets, Fog computing, cloud-based vehicular networks and multi-access edge computing.

The authors in [Taleb 2013c] propose their concept called *Follow-me-cloud* in

which services are migrating in conjunction with user movements taking into account various parameters such as latency, available bandwidth, server utilization. They provide as well an OpenFlow implementation [Taleb 2013a].

Many publications referring to these "Follow me" approaches exist [Ouyang 2018] [Aissioui 2018][Taleb 2013b].

Service migrations in Fog Computing infrastructure borrows ideas and combines techniques used in handovers in cellular networks and live migration in Cloud computing. A reader can refer to the survey [Wang 2018b] where the authors compare these two concepts with service migration in MEC as well their different scopes and similarities.

In [Li 2019] authors propose a quality-of-service aware scheme based on the existing handover procedures to support the real-time vehicular services. A case study based on a realistic vehicle mobility pattern for the Luxembourg scenario [Codecá 2017] is carried out, where the proposed scheme, as well as the benchmarks, are compared by analyzing latency and reliability as well as migration cost.

We can also mention that [Agarwal 2010] discuss Volley, an automatic service placement for geographically distributed DCs based on iterative optimization algorithms. Volley migrates services to new DCs if the capacity of a DC changes or the user changes location.

More theoretical results are given in [Taleb 2013b], [Urgaonkar 2015]. In the first study, the authors use Markov Decision Processes to capture the tradeoff between migration cost and user experience. In the second study, the authors focus on the similar problem. They model this tradeoff due to service migration's network overhead and latency for the use. Since service migration affect workload scheduling they tackle this decision jointly. Instead of using dynamic programming to solve MDPs, they decouple the MDPs and apply the technique of Lyapunov optimization of control theory.

In [Yao 2015] a contribution on service migration in the area of VANETS is given. The authors present a Mixed Integer Quadratic Programming formulation. As an input they have a graph of RSUs, resources, vehicles and sessions and they try to solve a minimum network cost VM migration problem. They then propose an heuristic algorithm with polynomial time.

Another theoretical work is [Gonçalves 2018], where authors propose a VM migration approach based on mobility prediction. The authors provide an ILP model for VM placement in Fog computing. The problem they are dealing with is different from ours. Their algorithm defines the set of candidate cloudlets to receive the user's VM according to the user's future position. In their model they want to maximize the accepted requests while minimizing the user's latency.

They execute these two objective functions sequentially.

We finally mention tools such as described in [Lopes 2017], which is an extension of the iFogSim simulator [Gupta 2016] adding virtual machine migration policies for mobile users. Their VM migration policy takes into account user's position, speed and direction. As well they define, the point where the migration can be potentially triggered by specifying the geographical zone based on user's coordinates. *FogNetSim* has been proposed in [Qayyum 2018]. *FogNetSim* is an extension of *OMNET++* that is used to simulate a network. The tool simulates among others geographically distributed data centers while providing support for handovers among Fog nodes. The implementation includes a number of mobility models. It is a release of 2018 and as future work authors mention the support for VM migration. [Rejiba 2019].

Our approach differs from these studies since it is the first trying to minimize the global number of migrations taking into account traffic sent across the Fog network.

4.3 System Model

Given a set of candidate locations for Fog clouds, we study the problem of obtaining the optimal set of Fog nodes. The optimality criterion is a linear combination of the cost of infrastructure and the number of migrations. The cost of infrastructure increases with the number Fog nodes.

In this problem, we have multiple users and we assume that the statistics of base station handovers are given. These statistics are used to compute the total number of migrations.

We define a cost that is a function of the number of migrations between base stations. In this model, all base stations are paired. Migrations happen between two base stations and incur a cost when they do not share a specific Fog node. We call this model pairwise because all cost components happen between two base stations and depend on the traffic between them. If there is no traffic, no migration is triggered. This model is a simple first approach to explore the general idea of optimizing service migrations using mobility patterns. Unfortunately it does not address a number of scenarios correctly. For instance, while adding more Fog nodes should naturally lead to more migrations, this model cancels them out and it results in a decreased cost.

An extended path-based approach that solves these issues is presented at the end of the chapter. However, if one only has handover statistics without being able to track users, this model provides a solution that encourages minimizing migrations.

Organization

The rest of this work is organized as follows. In Section 4.4.1, we describe the general model, and present an Integer Linear Program (ILP) formulation of the problem. In Section 4.4.2, we present a heuristic to compute a feasible solution to this ILP. This heuristic is based on the greedy algorithm for the weighted set cover problem. In Section 4.4.3, we evaluate the performance of the heuristic and compare it with the standard greedy algorithms for the set cover and weighted set cover problems.

4.4 Contributions

4.4.1 Pairwise mobility

In this section, we formulate the problem of minimizing the minimum number of service migrations between base stations as an ILP problem. By pairwise mobility model, we mean that the origin and the destination base stations are neighbors and the paths are direct routes from the origin to the destination. In the example in Figure 4.1, the pairwise mobility model has traffic going between only pairs of base stations, that is routes will be $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{B, A\}$, $\{B, C\}$, $\{B, D\}$ and so on.

In Figure 4.1, the migration model we have defined has the following attributes: for the pair (A, B) or (B, A) there will be no migration since both base stations A and B share Fog node 1. However for pairs such as the (A, C) (or (B, D), etc) there will be one migration since they share no common Fog node. What this model does not support, however, is computing that there will be one migration by going from A to B to C, because (A, B) share the Fog node 1, and (B, C) share the Fog node 2. Instead the model will consider there is no migration happening. In a more realistic setting, if the service to migrate is first on Fog node 1 (routed from base station A then B) and later migrated on Fog node 2 (routed from base station C), there would be a need for one migration. This specific model has insufficient input to map this. At the end of the chapter, a path-based approach is suggested to address this limitation.

Assume we are given a traffic mobility pattern between various base stations. This pattern is summarized in a matrix, W, whose (i, j)th element, $w_{i,j}$ represents the number of vehicles that move from base station i to base station j per unit time. We shall call W the mobility matrix.

We are also given a connectivity matrix C whose (i, j)th element, $c_{i,j}$, is 1 if data center j meets the SLA for traffic from base station i, and 0 otherwise.

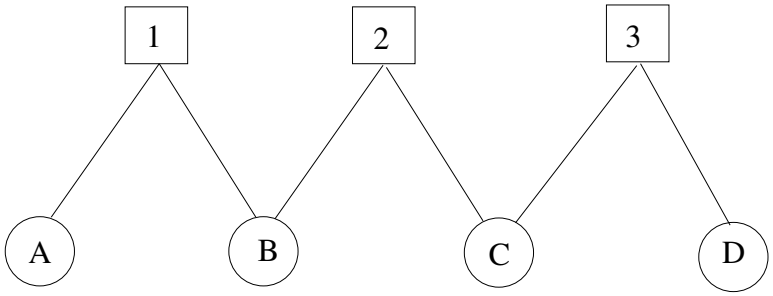


Figure 4.1: Example bipartite graph

We wish to determine the set of Fog nodes to which each base station should send its data traffic to. Implicitly, this determines which Fog node sites within the set $\mathcal{F}$ should be operational. The choice of operating or not a Fog node is influenced by two conflicting costs. First, there is a cost of operating a Fog node which includes the infrastructure as well as the maintenance costs. Thus, the larger the number of Fog nodes selected by one of more base stations, the larger the opening cost. The other cost is for service migration. When a vehicle moves out of the range of base station i and into the range of base station j, in order to maintain the continuity of its services, its data has to be migrated from a data center selected by i to one selected by j. This migration cost can be avoided if both i and j have a data center in common.

As we have already mentioned in the introduction what we aim to find is the vector x of Fog nodes that will operate and result in the minimum migration and operation cost.

Let $x_k \in \{0, 1\}$ be a binary variable that indicates if Fog node k is operational or not.

Definition of a migration

The following two relations are used to define a migration:

$$y_{i,k} = c_{i,k} x_k, \tag{4.1}$$

$$z_{i,j} = 1 - \max_k (y_{i,k} \cdot y_{j,k}) \tag{4.2}$$

- Equality (4.1) is used to specify which Fog nodes k serve the base station i under a SLA. If the Fog node k does not serve the base station i under a SLA, then $c_{i,j}$ will be zero and then $y_{i,k}$ is zero.

- In equality (4.2) $z_{i,j}$ indicates whether i and j have a node in common in the set of operational Fog nodes indicated by $\mathbf{x}$. It is used to count the number of migrations.

Number of migrations

The number of service migrations due to vehicles moving between i and j is given by

$$m_{i,j}(\mathbf{x}) = w_{i,j} z_{i,j} \tag{4.3}$$

Note that $m_{i,j}$ does not depend upon how many Fog nodes are in common between i and j as long as there is at least 1.

The optimization problem is formulated as the following Integer Program:

$$\underset{\mathbf{x}}{\text{minimize}}\ g(\mathbf{x}) + \beta \sum_{i,j} m_{i,j}(\mathbf{x}) \tag{4.4}$$

$$\text{s.t.}\quad \mathbf{x} \in \{0,1\}^{|\mathcal{D}|},$$

$$y_{i,k} = c_{i,k} x_k,\ \forall k \in \mathcal{D}, \tag{4.5}$$

$$\sum_k y_{i,k} \geq 1,\ \forall i. \tag{4.6}$$

In (4.4) the function g is the operational cost and could potentially depend upon which Fog nodes are open. In the simplest non-trivial case, $g(\mathbf{x}) = \sum_k x_k$, i.e. a linear cost of operating a Fog node. In our experiments we gave the coefficient β values such as 0.01 or 0.001 depending on how much weight we attribute to migrations, in addition in all our experiments the operation cost of one Fog node is 1. $g(x)$ thus corresponds to the number of Fog nodes. If there are three selected Fog nodes then $g(x) = 3$.

The migration cost in the objective function is defined as the sum of the resulting cost for each pair of base stations. The cost for a pair (i, j) of a base station is $(w_{i,j} + w_{j,i}) \cdot z_{i,j}$ where $z_{i,j}$ is 0 if (i, j) share a base station in x. It is 1 otherwise.

Finally, the constraint (4.6) expresses that every base station should be connected with at least one Fog node under an SLA.

Summing up, the two main specifications to take into consideration for the vehicle mobility are that (a) if two base stations have a Fog node in common and that Fog node is selected, then the cost for the migration between those two base stations is 0, (b) all base stations should be connected to at least one selected Fog node.

Table 4.1 presents all variables used.

Table 4.1: Variables used in the ILP formulation

Notation	Description
W	mobility matrix
C	connectivity matrix
x_k	A binary variable indicating whether Fog node k is operational or not
z_{ij}	A binary variable indicating whether i and j have a Fog node in common
w_{ij}	variable in the mobility matrix, represents the number of vehicles moving from base station i to base station j
m_{ij}	Number of service migration per unit time
c_{ij}	binary variable in the connectivity matrix, if 1 it means the j Fog node meets the SLA for service coming from i
$g(\mathbf{x})$	operational cost which depends upon which Fog nodes are open.

Linearising this model to use in Gurobi

The term $z_{i,j} = 1 - \max_k(y_{i,k} \cdot y_{j,k})$ is not linear due to the multiplication operation. Since Gurobi is a linear solver we alter the problem formulation in order to get a linear expression. Note that Gurobi can implement the maximum operator. So, we do not linearize it.

We first define variable $r_{i,j,k} = y_{i,k} \cdot y_{j,k}$. The expression $r_{i,j,k} = y_{i,k} \cdot y_{j,k}$ can be linearised using the following rules [Brown 2007].

$$\begin{aligned} r_{i,j,k} &\geq y_{i,k} + y_{j,k} - 1 \\ r_{i,j,k} &\leq y_{i,k} \\ r_{i,j,k} &\leq y_{j,k} \\ 0 \leq r_{i,j,k} &\leq 1 \end{aligned}$$

Now, $z_{i,j} = 1 - \max_k(y_{i,k} \cdot y_{j,k})$ expresses that a pair of base stations i and j

will account a cost $(w_{i,j} + w_{j_i}) \cdot z_{i,j}$.

$$\underset{\mathbf{x}}{\text{minimize}}\ g(\mathbf{x}) + \beta \sum_{i,j} w_{i,j} \cdot z_{i,j} \tag{4.7}$$

$$\text{s.t.}\quad \mathbf{x} \in \{0,1\}^{|\mathcal{D}|},$$

$$y_{i,k} = c_{i,k} x_k, \tag{4.8}$$

$$\sum_k y_{i,k} \geq 1, \forall i \tag{4.9}$$

$$r_{i,j,k} \geq y_{i,k} + y_{j,k} - 1 \tag{4.10}$$

$$r_{i,j,k} \leq y_{i,k}, r_{i,j,k} \leq y_{j,k}, r_{i,j,k} \in \{0,1\},\ \forall i,j,k, \tag{4.11}$$

$$z_{i,j} = 1 - \max_k(r_{i,j,k}),\ \forall i,j \tag{4.12}$$

4.4.2 Heuristics

The weighted Set Cover problem which has been presented in Section aims to find the subsets whose union covers all elements in the universe. In our problem formulation, the subsets in the weighted set cover are the individual Fog nodes serving the base stations under an SLA.

We modify the weighted set cover by using two heuristics cost functions: *(i)* a cost function depending on the amount of incoming traffic to each base station, and *(ii)* a cost function that depends only on the additional migrations.

Weights / Cost function

We transform the Fog nodes and base station connectivity bipartite graph into an instance of the Weighted Set Cover problem. The Fog nodes are represented as subsets containing the base stations they are connected to. The optimal Set Cover is thus the selection of the fewest Fog nodes that cover all base stations. Using Weighted Set Cover, we attribute weights to each subset (Fog node) and aim to find the combination of Fog nodes that cover all base stations with the minimum weight.

We give each Fog node a weight equal to the sum of incoming traffic to each base station it is connected to.

For instance in our example in Figure 4.2, we compute the traffic incoming for base station A first. There are 30 handovers from base station B to A and 50 handovers from C to A, for a total of 80 incoming handovers. Base station B incoming traffic consists in 10 handovers from A to B and 60 handovers from base station C to B, for a total of 70 incoming handovers. The Fog node 1 is

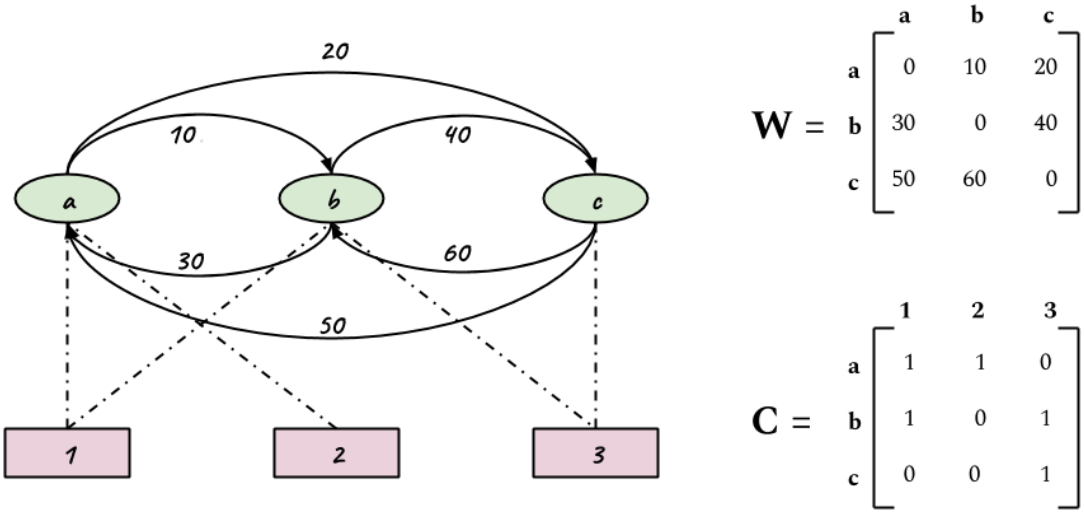


Figure 4.2: Pairwise mobility

serving both A and B and is thus mapped as a subset $\{A, B\}$ will have a cost of 150 $(80 + 70)$.

These weights attributed to each subset to let Weighted Set Cover prioritize Fog nodes more likely to have higher activity.

This is represented by the following weight function counting that sum of incoming traffic to each of base stations in a subset: Given W the mobility matrix, for a subset S, $\sum_{j \in S} \sum_{i \to j} w_i$. Our model wants to favor higher weights. Since the weighted set cover algorithm minimizes the weight we are taking the inverse of the weights.

Modified weighted set cover

We also tried to modify the weighted set cover algorithm by iterating until the universe is covered and at each step picking the Fog node that minimizes the number of additional migrations. The benefit of this modification compared to the default weighted set cover implementation is that it does not need to take the weights as an input. While we previously had to compute the weights by summing incoming traffic per base station, this modified algorithm selects the Fog nodes by computing their weight greedily (Algorithm 2).

For $j \in \mathcal{F}$, denote B_j, the set of base stations covered by j. With slight abuse of notation we will replace j by a subset of $\mathcal{F}$ in which case B_A will be the set of base stations covered by the subset A. Also, for $A \subset \mathcal{F}$, A^c will denote the

complement inside $\mathcal{F}$, that is, the set $\mathcal{F} \setminus A$.

For a set $A \subset D$ define:

$$\gamma_A(k) = \frac{\sum_{(i,j) \notin B_k \setminus B_A} m_{i,j}}{|B_k \setminus B_A|}, \ \forall k \notin A. \tag{4.13}$$

The ratio $\gamma_A(k)$ can be interpreted as additional migrations per base station that will be covered if k is added to A. A higher value of γ, indicates that this Fog node eliminates the need for a larger number of migrations and that it has a lower marginal cost.

Algorithm 2: Modified weighted set cover

```
Output: A
C ← ∅
A ← ∅
while C ≠ B do
    k̂ = arg min_{k∈A^c} γ_{A^c}(k)
    C ← C ∪ B_k̂
    A ∪ k̂
```

4.4.3 Performance Evaluation

In this section we describe the results obtained from three different heuristics: a plain set cover implementation, a weighted set cover implementation where weights represent the inverse of inbound traffic on each Fog node and a modified version of weighted set cover which does not start with explicit weights but computes them greedily. We also show the solution to ILP which was computed using Gurobi [Gurobi Optimization 2018] which is a solver for optimization problems.

In the first example, we show how the solution of the different heuristics varies with β which is the weight if the migrations. For Figure 4.3 we took the number of base stations $|\mathcal{B}| = 10$ and the number of Fog nodes $|\mathcal{F}| = 5$. Thus C is a 10×5 matrix and W is a 10×10 matrix. The number of cars was between 0 and 100 in the mobility matrix W.

When β is close to 0, the ILP is almost like a set cover problem as the cost of migrations is not taken into account in the optimization. For larger values of β, the migrations become important and the greedy algorithm for set cover becomes worse.

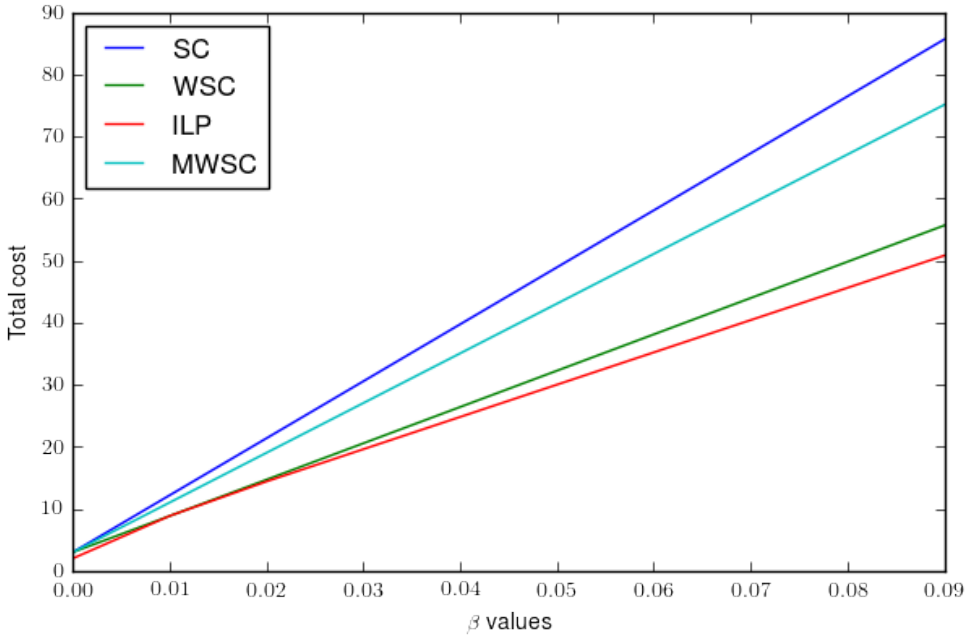


Figure 4.3: Cost as a function of β for Set Cover (SC), Weighted Set Cover (WSC), Modified Weighted Set Cover (MWSC) and the ILP formulation.

In the second set of experiments we increase the number of base stations and fog nodes. We take different pairs of $(\mathcal{B}, \mathcal{F})$ with maximum $|\mathcal{B}| = 50$ and maximum $|\mathcal{F}| = 15$. The traffic matrix has random entries between 0 and 20, and 100 different random matrices were generated. For $\beta = 0.01$, the average total cost is shown in Fig. 4.4 and the average execution time is shown in Fig. 4.5.

For $\beta = 0.001$, the results are shown in Fig. 4.6 and 4.7. Again for smaller values of β, the heuristics are close to ILP but with a smaller execution time than ILP.

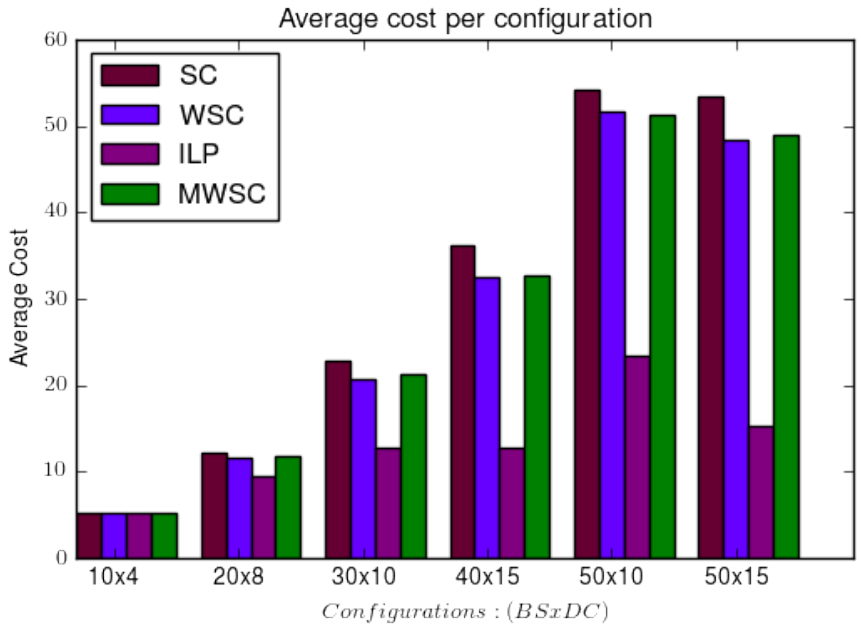


Figure 4.4: Average Cost of the SC, WSC, MWSC and ILP given $\beta = 0.01$ for 50 base stations and 20 fog nodes

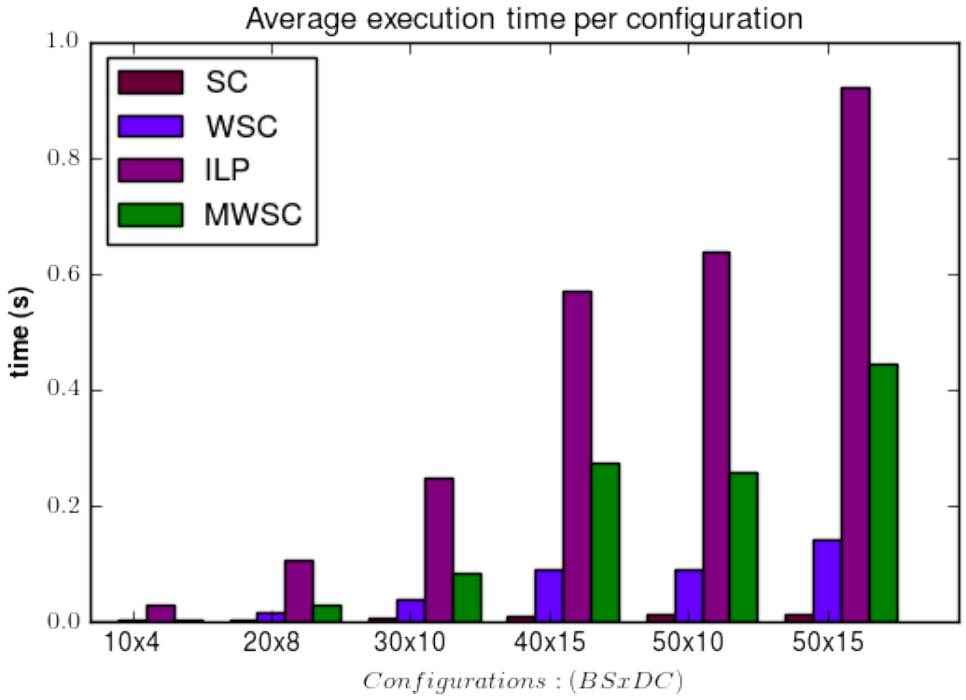


Figure 4.5: Execution times of the SC, WSC, MWSC and ILP given $\beta = 0.01$ for 50 base stations and 20 fog nodes

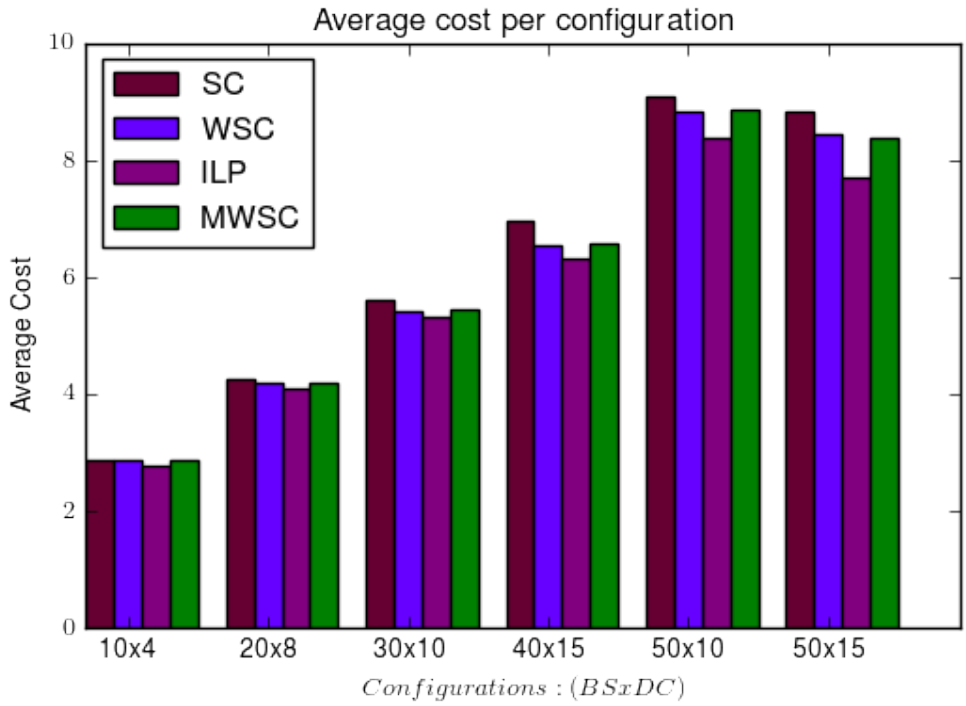


Figure 4.6: Average Cost of the SC, WSC, MWSC and ILP given $\beta = 0.001$ for 50 base stations and 20 fog nodes

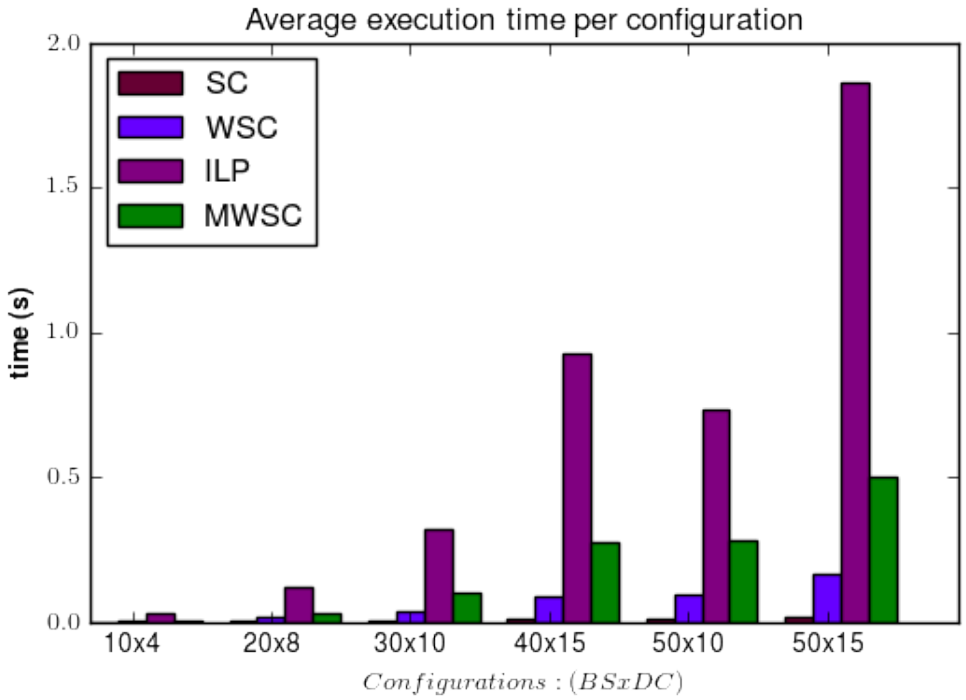


Figure 4.7: Execution times of SC, WSC, MWSC and ILP given $\beta = 0.001$ for 50 base stations and 20 fog nodes

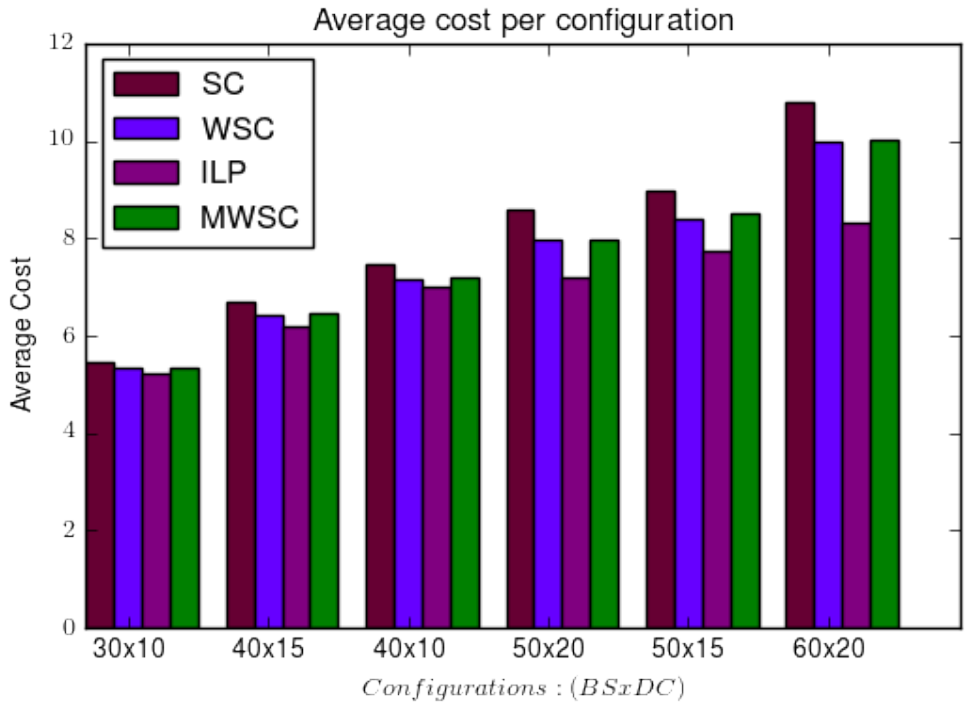


Figure 4.8: Average cost of SC, WSC, MWSC and ILP given $\beta = 0.001$ for 60 base stations and 20 fog nodes

Finally, in Fig. 4.8 and 4.9 we plot the results for maximum $|\mathcal{B}| = 60$ and maximum $|\mathcal{F}| = 20$. The traffic matrix has random entries between 0 and 20, and 100 different random matrices were generated, and $\beta = 0.001$

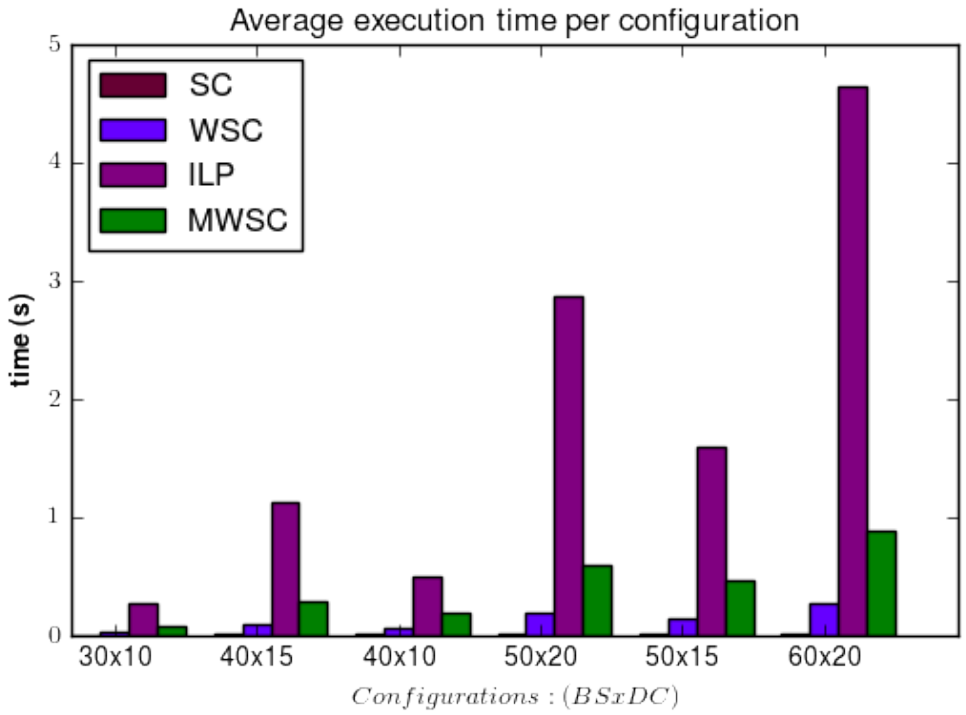


Figure 4.9: Execution times for a $\beta = 0.001$ for 60 base stations and 20 fog nodes

We expect the traffic statistics to be updated regularly, without necessarily requiring the changes to happen in real-time. Depending on how often operators would want to update the traffic data, they could either pick the faster heuristic or spend the extra compute time with a more accurate solution provided by an ILP solver.

4.5 Conclusion and Future Work

We have presented an Integer Linear Program for obtaining the optimal location of fog nodes that minimizes a linear combination of the number of migrations and the cost of infrastructure. This was done for a pairwise mobility model in which the origin and destination base stations are neighbors. We gave two heuristics based on the Weighted Set Cover problem and evaluated the performance of these heuristics with that of the optimal solution. This study focuses on pairwise base station handovers which are a limited form of mobility patterns. It is interesting to extend this model by tracking the path taken between base stations by indi-

vidual users to gather frequency statistics. Given this input we can imagine more accurate models to minimize service migrations in a way that supports better precise mobility pattern. In the following section which is proposed as a future work we explain this idea.

4.5.1 Path-based Mobility Patterns

In this section we discuss the extension of our model in the case of having statistics over itineraries for a given area. To illustrate this more clearly let us define a path in the road network by a set of ordered base stations. We shall assume we are given as input the traffic statistics of how many users move along each path 4.10.

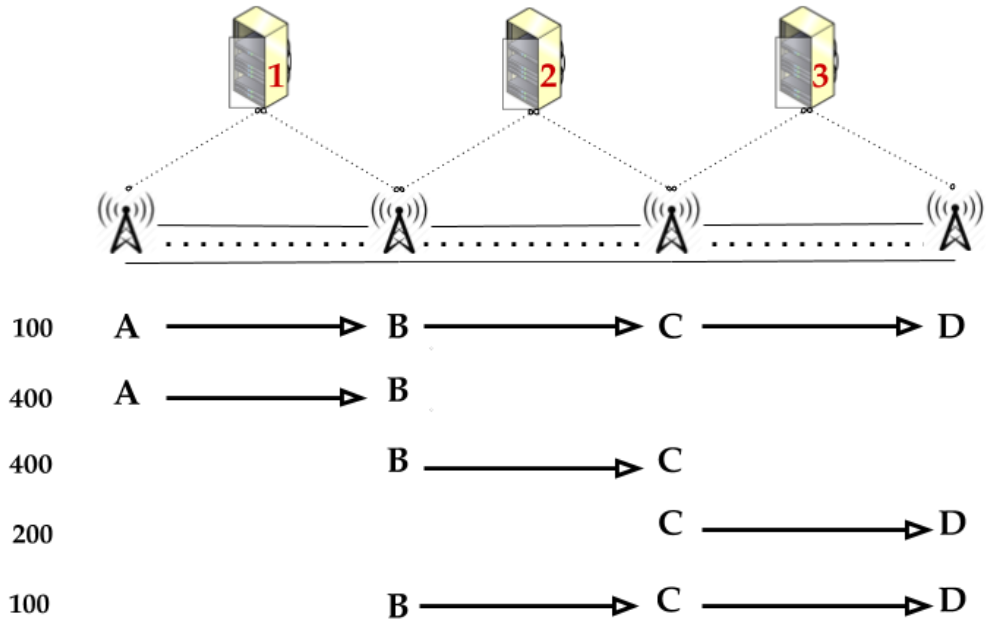


Figure 4.10: Path-based mobility

As an example, consider the network in Figure 4.1 with four base stations $\{A, B, C, D\}$ and three fog nodes $1, 2, 3$. Here $F_A = \{1\}$, $F_B = \{1, 2\}$, $F_C = \{2, 3\}$ and $F_D = \{3\}$. There are only two subsets of $\mathcal{F}$ that can cover $\mathcal{B}$: $\{1, 3\}$ and $\{1, 2, 3\}$.

Suppose there is only one path $\{A, B, C, D\}$ in the network with 100 users

Path	$A \to 1$, $B \to 1$, $C \to 3$, $D \to 3$	$A \to 1$, $B \to 1$, $C \to 2$, $D \to 3$	$A \to 1$, $B \to 2$, $C \to 2$, $D \to 3$
100	100	200	400
400	0	0	400
400	400	400	0
200	0	200	200
100	100	200	100
Total	**600**	**1000**	**1100**

Table 4.2: Migrations costs for the different configurations

taking this path. Then, the number of migrations for $\{1, 3\}$ is 100 (one for each user migrating from fog node 1 to 3 when the user moves from base station B to base station C) whereas for $\{1, 2, 3\}$ this number is again 100 (assuming 2 is not used) or 200 (if we assume that A is connected to 1, B and C are connected to 2 and D is connected to 3). Assuming the infrastructure cost is linear in the number of fog nodes, then it is optimal in this case to operate fog nodes 1 and 3 only.

Now, suppose that in addition to the previous users there are 500 users that take the path $\{B, C\}$ only. Now, the number of migrations for $\{1, 3\}$ is 600 whereas for $\{1, 2, 3\}$ it is $200 \times 1 + 500 \times 0 = 100$ (this is because the users on the path $\{B, C\}$ will be connected to fog node 2 and hence will not require any migrations). It might be more profitable in this case to operate the set $\{1, 2, 3\}$. This will depend upon the cost of infrastructure.

In Figure 4.10 we provide a richer example with five different paths with different amount of traffic. We compare three different routing strategies ($A \to 1$, $B \to 1$, $C \to 3$, $D \to 3$, $A \to 1$, $B \to 1$, $C \to 2$, $D \to 3$ and $A \to 1$, $B \to 2$, $C \to 2$, $D \to 3$) and calculate how many migrations are required for each path depending on the routing strategy. We can then select the routing strategy that minimizes the overall amount of migrations, in this case the strategy from the first column will trigger only 600 migrations as opposed to 1000 and 1100 for the second and third strategies. Table 4.2 details the different steps.

This problem can be solved using Gurobi or with a greedy heuristic.

Chapter 5

Adaptive Task Allocation in the Fog

In this chapter, we propose a simulation model to compare different task allocation strategies for Fog Computing. While we focused on the design phase in the previous two chapters, we now examine solutions for the operation phase: even if the infrastructure has been optimally planned, we know traffic and workload characteristics will deviate from expectations. We thus need, on top of a good design, to devise efficient resource management mechanisms that adapt to these deviations. The simulation model we propose relies on the network simulation framework OMNeT++.

We focus on online distributed adaptive task allocation mechanisms which require neither cooperation between base stations nor knowledge of the physical infrastructure. We assume that each base station independently learns what is the optimal task allocation, without coordination or even clock synchronisation with the other base stations and just reacts to the response times it observes from the Fog nodes.

In our experiments we tune different parameters of our network such as the communication time between base stations and Fog nodes, the processing capacity of micro-datacenters or the distribution of service times in the Fog and the Cloud. For instance we assume exponential service distributions in some scenarios and in others we assume the processing times of the jobs follow a Pareto distribution to test the robustness of different allocation schemes by observing the increase in the mean response time.

We also add perturbations after some time in the experiment to see how the different strategies react. In all these cases we provide extensive performance comparisons.

5.1 Motivation

In Chapter 3, we have devised an algorithm for designing a Fog Computing infrastructure, that is, for deciding where to place Fog nodes, how much capacity to install in each of them, and how to route the flows of jobs originating from the base stations. The proposed algorithm is based on a very simple queuing model of Fog nodes and assumes fixed communication delays. The algorithm also assumes that the intensity of the traffic originating from base stations is exactly predictable. In practice, these assumptions are of course not perfectly met. They can be seen as a blessing in disguise which makes the problem tractable and do not affect too much the design placement and capacity planning decisions.

However, in the exploitation phase, the blind use of the task allocation strategy obtained in the design phase may lead to poor network performances. Indeed, this strategy was obtained using an overly simple model. It is also likely that at some point in time the actual traffic matrix deviates significantly from the one used for designing the infrastructure. A well-known alternative approach is to rely on online adaptive task allocation mechanisms. In this chapter, we shall particularly focus on distributed task allocation algorithms which requires neither cooperation between base stations nor knowledge of the physical infrastructure. More precisely, we consider learning-based mechanisms which do not rely on a specific model of the infrastructure but just react to the response times they observe from the Fog nodes. We assume that each base station independently learns what is the optimal task allocation, without coordination or even clock synchronisation with the other base stations.

We adopt a discrete-event simulation approach. We propose a simulation model of a Fog Computing infrastructure which relies on the network simulation framework OMNeT++. Several adaptive task allocation schemes are implemented by the simulation model and we study several scenarios to understand the advantages and drawbacks of each of them.

This chapter is organized as follows. Section 5.2 is devoted to related work. We present the simulation model in Section 5.3. We first give a short overview of the simulation framework OMNeT++ in Section 5.3.1, and describe our simulation model in Section 5.3.2. Section 5.4 presents the different task allocation strategies considered in this chapter. Numerical results are presented in Section 5.5. Finally, some conclusions are drawn in Section 5.6.

5.2 Related literature

In our study we examine the effect of various task allocation strategies using different configurations that let us compare the performance of Fog allocation, Cloud allocation and offloading.

An important contribution that inspired this work is [Wang 2018a]. It aims to allocate tasks adaptively and efficiently in the Cloud and provides a platform named TAP (Task Allocation Platform) that uses a Linux kernel module and can compare scenarios in a realistic setting. In our study we use an Omnet++ simulation model instead. In comparison with our work this happens in a centralized fashion since the TAP controller gathers all information. The sensible strategy we use is taken from this paper and we compare it with many more strategies.

Routing is another application where adaptive algorithms are commonly used. In [Jonglez 2016] the authors propose a decentralized algorithm robust to measurement errors, outdated information and clock desynchronization that converges to a pure Nash equilibrium. They use the Mininet network emulator to test the behavior of their implementation in different scenarios. The Optimal Path Selection (OPS) algorithm is a probabilistic algorithm for adaptive single-path routing in packet-switched networks [Jonglez 2017]. It was developed in the context of atomic non-splittable routing games and is inspired from a mirror-descent algorithm for general potential games [Coucheney 2015].

In [Beraldi 2017] the authors formulate a cooperative offloading policy between two edge data centers for load balancing. They define a blocking state in which the requests are dropped and compare with two other schemes in order to minimize the amount of blocked requests: with an isolated policy, where no data center works with cooperation with the other, and a fully shared where any request is forwarded to any other datacenter. The cooperative scheme they propose behaves better than the two others. The problem the authors study is similar with ours as the offloading scenario we study dynamically happens when a Fog node is overloaded with tasks. However their scheme is static whereas the strength of our approach is that it is dynamic and can adapt to the unknown.

A theoretical work is [Fricker 2016] where authors consider the scenario of offloading with a certain probability blocked requests at the Fog to the neighboring data centers and to the Cloud. Through functional equations and Markov theory they estimate the gain achieved via cooperation between neighboring data centers.

In [Baek 2019], the authors formulate the load balancing problem as a Markov Desicion Process (MDP) solved by Q-learning. Q-learning is a common reinforcement learning algorithm that can solve MDPs. The multi-armed bandits

algorithms we use are simpler forms of reinforcement learning that deal with problems with only one state in contrast with MDPs that model problems with several states. This allows Q-learning models to keep knowledge over time and remember the past to better handle the future. Their reward function takes into account processing time to minimize and overload probability. A difference in our approach is that we take the constraint of giving base stations no knowledge of the system, while the authors place their Q-learning algorithm in the SDN controller which has an overview of all the system.

The authors in [Wang 2018a] use a reinforcement learning algorithm using a random neural network proposed in [Gelenbe 1989] in order to allocate tasks to the Cloud.

There are different studies on offloading which describe benefits other than latencies improvements, such as energy saving opportunities. This is particularly interesting in fog models where Fog nodes may be battery-powered devices. In [Liu 2018] for instance authors formulate a multi-objective optimization problem to minimize the energy consumption, execution delay and payment costs that finds the optimal probability to offload.

We can mention existing surveys on computation offloading such as [Cheng 2019] [Lin 2019] which provide a review of the state-of-the-art of computation offloading in the various contexts it can be useful: energy consumption minimization, Quality of Services guarantees, and computation and storage requirements. There are also a number of tools and frameworks that help building offloading infrastructure or mobile application developers implement the required facilities. In [Cuervo 2010] the authors present a framework that allows computations to be dynamically offloaded to the Cloud with a runtime optimizer that can transform local computation into remote calls on-demand and thus make it easy for application developers to built adaptive applications. They show energy consumption improvement in resource-intensive workloads.

5.3 OMNeT++-based Simulation Model

5.3.1 The Network Simulation Framework OMNeT++

OMNeT++ (Objective Modular Network Testbed in C++) is a C++ framework which provides infrastructure and tools for writing discrete-event simulations. It was designed to be as general as possible working for a number of use cases in communication networks such as wireless and ad-hoc network simulations, peer-to-peer networks, queuing network simulations, etc [Varga 2008]. One of the fundamental ingredients of this infrastructure is a component architecture

for simulation models. Models are assembled from reusable components called *modules*. Modules can be connected with each other via *gates*, and combined to form compound modules (see Fig. 5.1). They communicate through message passing, where *messages* may carry arbitrary data structures.

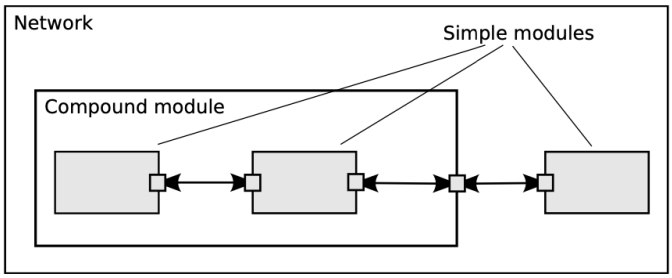


Figure 5.1: Nested modules.

The behavior of the model is defined in C++, while the structure of a simulation model is described in the NED language, OMNeT++'s topology description language,which stands for Network Description. NED allows the user to connect C++ modules. In the following sections we introduce the NED language, as well some basic components of OMNeT++ in order to provide the necessary background about our Fog tool and configurations.

The NED language

The user describes the structure of a simulation model in the NED language. NED lets the user declare simple modules, and connect and assemble them into compound modules. The user can label some compound modules as networks; that is, self-contained simulation models. Channels are another component type, whose instances can also be used in compound modules.

We introduce the NED language via a simple example. At the top, we define the network model as follows.

Listing 5.1: NED file defining the network model.

```
1
2 network Network
3 {
4   submodules:
5     node1: Node;
6     node2: Node;
7     node3: Node;
8     ...
9
```

```
10    connections:
11      node1.port++ <-->  {datarate=100Mbps;}  <--> node2.port++;
12      node2.port++ <-->  {datarate=100Mbps;}  <--> node4.port++;
13      node4.port++ <-->  {datarate=100Mbps;}  <--> node6.port++;
14      ...
15  }
```

The network contains several nodes, named `node1, node2`, etc. from the NED module type `Node`. We shall shortly define `Node` below.

The second half of the declaration defines how the nodes are to be connected. The double arrow means bidirectional connection. The connection points of modules are called gates, and the `port++` notation adds a new gate to the `port[]` gate vector. We can as well specify properties for the connection such as latencies or datarates.

Omnetpp.ini file

The central configuration point of the OMNeT++ simulation environment that let us define the parameters and how the simulation behaves with different inputs is the omnetpp.ini file. One can define any number of networks in the NED files, and for every simulation the user has to specify which network to set up. The usual way of specifying the network is to put the network option into the configuration file. Examples of content of configuration files will be presented in the next section.

5.3.2 Discrete Event Fog Simulation Model

We describe our simulation model using a simple scenario which is illustrated in Fig. 5.2. This scenario is composed of two base stations sending execution requests to two different Fog nodes.

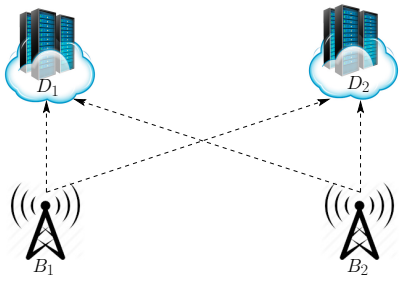


Figure 5.2: A simple scenario.

In our simulation model, this scenario can be described with the NED file given in Listing 5.2. In this network, the nodes `client[0]` and `client[1]` are of type `AccessNode` and represent the base stations B_1 and B_2 in Fig. 5.2, whereas the nodes `fog[0]` and `fog[1]` are of type `FogNode` and represent the Fog nodes D_1 and D_2 in Fig. 5.2. For simplicity, we assume here that all communication delays are drawn from a truncated normal distribution, but it is of course possible to use asymmetric communication delays for the different connections.

Listing 5.2: The NED file for our simple scenario.

```
network Fog
{
  types:
        channel Channel extends ned.DelayChannel {
            delay = truncnormal(20ms,5ms);
        }

    submodules:
        client[2]: AccessNode;
        fog[2]: FogNode;

    connections:
      client[0].port++ <--> Channel <--> fog[0].port++;
      client[0].port++ <--> Channel <--> fog[1].port++;
      client[1].port++ <--> Channel <--> fog[0].port++;
      client[1].port++ <--> Channel <--> fog[1].port++;
}
```

We describe below the main components of our simulation model, including the simple modules `FogNode` and `AccessNode`.

A `Job` in the Fog Simulation Model

Messages are a central concept in OMNeT++. In our simulation model, message objects are used to represent jobs, but also events such as the end of service of a job on a given server. In our model a job is specified by a *source*, a *destination* and a *delay*, which represents the processing time of the job.

```
message Job
{
    int        source;
    int        destination;
    simtime_t  delay;
}
```

The simple module `FogNode`

The simple module `FogNode` is used to model a Fog node as a set of parallel queues. A Fog node has a certain number of `inout` gates that connect it to all the base stations. As described in Listing 5.3, the parameters of a Fog node are the number of servers it is composed of, the service time of a job on a server (for simplicity, we assume homogeneous servers) as well as the routing algorithm for assigning jobs to servers. Three routing policies are implemented: the *random routing policy* which assigns an incoming job to a server selected at random according to a uniform distribution, the *round-robin routing policy*, the *power-of-two-choices* and the *join-the-shortest-queue* policy which assigns the job to the server with the least number of jobs. The `meanServiceTime` represents the mean value of the service time. The parameter `deltaTime` gives the time interval between two consecutive recordings of the number of jobs in the system. Finally the last parameter `offloadthr` will be used for the offloading scenarios and as the name indicates provides a threshold after which the job is offloaded to the cloud.

Listing 5.3: The NED file of the simple module `FogNode`.

```
simple FogNode
{
    parameters:
        int              nbServers = default(1);
        double           meanServiceTime @unit(s);
        volatile double  serviceTime @unit(s);
        string           routingAlgorithm = default("random");
        double           deltaTime @unit(s) = default(0.1s);
        double           offloadthr  @unit(s) = default(1.0e6s);

    gates:
        inout port[];
}
```

The `serviceTime` is a volatile parameter which can be initialized in the `omnetpp.ini` file as follows:

```
Fog.fog[0].serviceTime = exponential(9ms)
Fog.fog[1].serviceTime = exponential(7ms)
```

In our simulations in order to measure the impact of the variability on service times we use a Pareto distribution instead of an exponential. In the `omnetpp.ini` file this is configured as follows:

```
Fog.fog[0].serviceTime = pareto_shifted(2.25,8.333ms,0.0)
Fog.fog[1].serviceTime = pareto_shifted(2.25,8.333ms,0.0)
```

The simple module `AccessNode`

The simple module `AccessNode` is used to model a base station. An access node has a certain number of `inout` gates that connect it to all the Fog nodes. As described in Listing 5.4. Each base station has a task allocation strategy that can be any of the strategies we have defined and assigns a routing probability to each Fog node it is connected to. There are other parameters such as inter-arrival time between job requests and traffic perturbations.

Listing 5.4: The NED file of the simple module `AccessNode`.

```
simple AccessNode
{
    parameters:
        volatile double interArrivalTime @unit(s);
        string jobName = default("job");
        string taskAllocation = default("sensible_routing");
        double deltaTime @unit(s) = default(0.2s);
        double startTime @unit(s) = default(-1s);
        double stopTime @unit(s) = default(-1s);
        double coefficient = default(2.0);

    gates:
        inout port[];
}
```

Traffic perturbations have three parameters : *start time* when perturbations should start, *stop time* when perturbations should stop and a coefficient. The coefficient is used as a mean to divide all inter-arrival times between the start time and the end time of the perturbation. Traffic perturbations are described in the `omnetpp.ini` file as follows:

```
Fog.client[0].startTime = 15s
Fog.client[0].stopTime = 30s
Fog.client[0].coefficient = 2.0
```

The above lines are used to divide all inter-arrival times by a factor 2 between the start time (at 15 s) and the end time (at 30 s) of the perturbation. Note that if the job arrival process is a Poisson process, this yields another Poisson process with a rate multiplied by 2. Indeed, if X is an exponentially distributed random variable with rate λ, then the random variable $Y = \frac{1}{k} X$ has a PDF given by

$$F_Y(x) = \mathbb{P}[Y \leq x] = \mathbb{P}[X \leq k\,x] = 1 - e^{-(k\lambda)\,x}, \quad \forall x \geq 0.$$

Job inter-arrival times are specified in the `omnetpp.ini` file:

```
Fog.client[0].interArrivalTime = exponential(12.5ms)
Fog.client[1].interArrivalTime = exponential(6.25ms)
```

After a new job reply is received from server `j` the router has the opportunity to update the routing probabilities and its state. These routing probabilities change across the different task allocation algorithm used. These adaptive algorithms are presented in the next sections.

5.4 Adaptive Task Allocation algorithms

In the following sections we present various algorithms that can be used in task allocation from base stations to Fog and Cloud nodes. These base stations have no knowledge of the infrastructure and they do not cooperate with each other. In one hand, this reduces the cooperation overhead, on the other hand it prevents base stations from sharing strategies.

Firstly we introduce static algorithms where there is no dynamic adaptation in order to use them as a base line. Bernoulli routing is one such example. We then present a number of adaptive learning based algorithms. One of them is *sensible routing*, described in [Gelenbe 2012], which has also been used to allocate tasks in the Cloud. The other adaptive algorithms we present are mostly taken from the theory of multi-armed bandits.

Multi-armed bandits [Lattimore 2020] is an important category of task allocations problems. These are a simple forms of reinforcement learning problems with a single state. They are used to evaluate the best choice in allocating a fixed quantity of a limited resource within a finite number of competing options.

The name comes from slot machines in gambling also called one-armed bandits. The gambler wants to play the best machine that maximizes his rewards. There are multiple one-armed bandits but the name is simplified as multi-armed bandits. The arms of the bandits in our case are the Fog nodes and the Cloud to which base stations want to allocate their tasks.

One important consideration of these problems is the exploitation versus exploration trade off. In our simulation model, we run simulations using three basic algorithms trying to deal with this trade-off: *ϵ-greedy*, *softmax* and *exp3* algorithms.

5.4.1 Static Task Allocation

In this task allocation strategy, which is also known as Bernoulli routing, an access node $i = 1, \ldots, K$ is given a fixed probabilistic choice vector $\mathbf{p}_i$, and allocates a job to fog node j with probability $p_{i,j}$, independently of the response times observed for previous jobs from this fog node and other fog nodes. A special case is the random task allocation strategy, which chooses the Fog node to which

a job request is assigned using a uniform distribution, that is, the probability that Fog node k is selected is $\frac{1}{N}$, where N is the total number of Fog nodes. Another special case that we shall consider in the following is the case where $p_{i,j} = 1$ if $j = j^*(i)$ and 0 otherwise, where $j^*(i)$ is the geographically closest micro-datacenter to the access node i.

5.4.2 Sensible Routing

The sensible routing algorithm was proposed in [Wang 2018a]. In this approach, the task allocation agent maintains a weighted average G_i of the response times of Fog node i. G_i is estimated for each of the Fog nodes i, and updated each time a new job reply is received. If d_i is the response time of the job received from Fog node i, then the value G_i is updated as follows

$$G_i \leftarrow (1 - \alpha)\, G_i + \alpha d_i,$$

where the parameter $0 \leq \alpha \leq 1$ is used to vary the weight given to the most recent measurement as compared to past values. In our experiment, we have used the value $\alpha = 0.1$. The probability p_i to allocate a job to Fog node i is then computed as follows

$$p_i = \frac{1/G_i}{\sum_{j=1}^{N} 1/G_j},$$

where N is the total number of Fog nodes. Of course, when a job has to be allocated, we use the most recent value of p_i which is available.

5.4.3 ϵ-Greedy Task Allocation

The ϵ-greedy algorithm is a method used for solving multi-armed bandits problem (see Chapter 2 of [Sutton 2018]). The basic idea of this method is to behave greedily most of the time, that is to choose the fog node with the lowest estimated response time most of the times, but every once in a while (with small probability ϵ) to select instead another fog node uniformly at random.

The task allocation agent maintains a weighted average of the response times of Fog node i. Let $G_i(k)$ be the estimated value of the response time of Fog nodes i after k job replies have been received from it. This estimated value is updated each time a new job reply is received as follows

$$G_i(k) = (1 - \alpha)\, G_i(k-1) + \alpha d_i(k),\ k = 1, 2, \ldots,$$

where $d_i(k)$ is the response time of the k^{th} job sent to Fog node i. The parameter $0 \leq \alpha \leq 1$ is used to vary the weight given to the most recent measurement as compared to past values. Initial values $G_i(0)$ are set to 0 to encourage exploration. Note that after k measurements, we have

$$G_i(k) = (1-\alpha)^k G_i(0) + \sum_{n=1}^{k} \alpha(1-\alpha)^{k-n} d_i(n),$$

so that the weight of a measure decays exponentially according to the exponent on $(1-\alpha)$.

Instead of choosing a constant step-size α, it is of course possible to vary the step-size as the number of measures grows. For stationary problems, it is known that the conditions $\sum_{k=1}^{\infty} \alpha_k = \infty$ and $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$ are sufficient to guarantee convergence to the (true) mean values. In particular, the choice $\alpha_k = 1/k$ is often used and gives

$$G_i(k) = \frac{\sum_{n=1}^{k} d_i(n)}{k},$$

that is, the estimated value of fog node i is just the average value of response times for jobs processed at this node. In our work, we shall however stick to a constant step-size α because it is known to be more convenient for non-stationary problems [Sutton 2018].

Assume that at time t (i.e. after t job replies have been received), fog node i_t^* is the one with the lowest estimated response time, that is, $i_t^* \in \text{argmin}_i G_i(t)$. Then, the next job is allocated to fog node i_t^* with probability $1-\epsilon$, and to another fog node $j \neq i_t^*$ with probability $\frac{\epsilon}{N-1}$, where N is the total number of Fog nodes.

5.4.4 Softmax Task Allocation

As the ϵ-greedy algorithm described above, the Softmax algorithm maintains an estimate $G_i(k)$ of the response time of Fog node $i = 1, \ldots, N$. However, the two algorithms differ in the way these estimates are used for task allocation. With the Softmax algorithm, the next job is allocated to Fog node i with probability

$$p_i = \frac{e^{-G_i(k)/\tau}}{e^{\sum_{n=1}^{N} -G_n(k)/\tau}},$$

where τ is a positive parameter called the temperature. High temperatures cause all choices to be (nearly) equiprobable, while low temperatures favor Fog nodes with low estimated response times (in the limit $\tau \to 0$, the Fog node with the

lowest estimated response time is selected with probability 1). In our experiments, we have used the value $\tau = 30$ ms, which is the value of the same order of magnitude as the total response time of a job.

5.4.5 EXP3 Algorithm

The *adversarial bandit* is a version of the multi-armed bandit problem introduced by Auer and Cesa-Bianchi in 1998 in which almost nothing is assumed about the mechanism that generates the rewards. In this problem, it is simply assumed that, at each iteration, an agent chooses an arm and an adversary simultaneously chooses the payoff structure for each arm. The goal is still to compete with the best action in hindsight. The EXP3 algorithm, which is given in Algorithm 3, was proposed in 2001 by Auer, Cesa-Bianchi, Freund, and Schapire [Auer 2002] and is known to have an expected regret bound of $\sqrt{2Tn\log(n)}$.

Algorithm 3: EXP3 algorithm.

Initialisation: $w_i(t) = 1$ for $i = 1, \ldots, N$.
for all $t = 1, 2, \ldots$ **do**
  Set $p_i(t) = (1-\gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \gamma\frac{1}{K}, \quad i = 1, \ldots, N$
  Draw next action j randomly according to the probabilities $p_1(t), \ldots, p_N(t)$
  Receive reward $x_j(t) \in [0, 1]$
  Set $w_j(t+1) = w_j(t) \times exp\left(\alpha\frac{x_j(t)}{Kp_j(t)}\right)$
end for

In the next section we present various experiments we performed using Static allocation, Sensible routing, Exp3, and Softmax.

5.5 Performance Evaluation

The objective of this section is to compare those adaptive learning-based task allocation schemes:

- We investigate whether the task allocation algorithms may lead to instabilities and routing oscillations.
- We also compare the convergence times of the various algorithms, and in particular, the speed at which they adapt to traffic perturbations.

- We assess the robustness of the various algorithms to measurement noises, such as highly-variable communication delays or service times.

We focus on two main scenarios. One scenario where base stations are connected to two Fog nodes and the main Cloud and can allocate their jobs without coordination. The second is similar but this time the Fog nodes are connected to the Cloud and can use a dispatching strategy to offload their tasks. We compare the two scenarios and the gains observed in response times by using offloading.

The organization of the experiments is the following. In the first scenario we perform experiments with constant link delays and with variable link delays. We as well compare the response times obtained for exponentially-distributed service times against those obtained with Pareto service time distributions. We then perform robustness tests by executing each scenario 5000 times and taking the average, minimal and maximal values on all of these scenarios. In the case of offloading, we first compare first four different dispatching policies: power-of-two-choices, join the shortest queue first, random policy and round robin and then we perform similar tests.

5.5.1 Task Allocation without offloading

A. Exponentially-distributed Service Times - Constant Link delays

We first consider the scenario illustrated by Figure 5.3. In this scenario, there are two Fog nodes (corresponding to `fog[0]` and `fog[1]`), one cloud datacenter (corresponding to `fog[2]`) and two access nodes. Note that communication delays are symmetric for `client[0]` and `client[1]`. The access node `client[0]` is closer to `fog[0]`, and similarly `client[1]` is closer to `fog[1]`.

The network is simulated for 100 seconds. As shown in Listing 5.5, each Fog node has only 5 parallel servers[1], and the service times are exponentially distributed with a mean of 15 ms for both nodes. In contrast, the Cloud (that is, `fog[2]`) has 100 parallel servers, and the service times in the Cloud are exponentially distributed with a mean of 10 ms. Jobs requests are generated according to Poisson processes, with a mean of 7.75 ms for the first access node and with a mean of 6.06 for the second one. The intensity of the traffic generated by the first access node is multiplied by 2.5 between $t_0 = 15$ s and $t_2 = 40$ s. The inter-arrival times of job requests at the second access nodes are divided by a factor 2.0 between times $t_1 = 30$ s and $t_3 = 60$ s. Note that, if we choose to always process an incoming job at the closest Fog node, it means that the utilization rate of

[1]The number of servers in Fog nodes and in the Cloud are of course not realistic. These values have been chosen so as to reduce the simulation times.

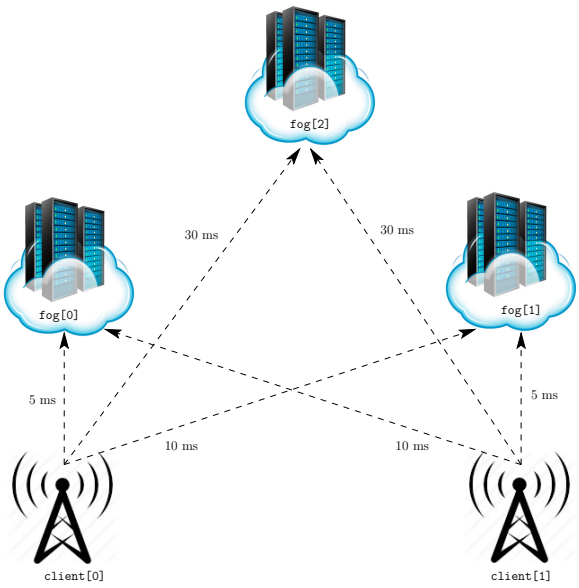


Figure 5.3: A simple scenario with two Fog nodes and one Cloud.

node `fog[0]` (resp. `fog[1]`), which is initially 0.387 (resp. 0.495), becomes 0.968 (resp. 0.99) between times t_0 and t_2 (resp. t_1 and t_3). In other words, under this routing strategy, the system is stable but operates in heavy load between times t_0 and t_3. Values are averaged over $5,000$ parallel simulation runs.

Listing 5.5: Parameters of the first scenario.

```
[Config FogCloudSim]
description = "Fog nodes serving jobs generated by Base Stations"
network = FogCloud
sim-time-limit = 100s
FogCloud.fog[0].serviceTime = exponential(15ms)
FogCloud.fog[1].serviceTime = exponential(15ms)
FogCloud.fog[2].serviceTime = exponential(10ms)
FogCloud.fog[0].nbServers = 5
FogCloud.fog[1].nbServers = 5
FogCloud.fog[2].nbServers = 100
FogCloud.client[0].interArrivalTime = exponential(7.75ms)
FogCloud.client[1].interArrivalTime = exponential(6.06ms)
**.routingAlgorithm = "random"
FogCloud.client[0].startTime = 15s
FogCloud.client[0].stopTime = 40s
FogCloud.client[0].coefficient = 2.5
FogCloud.client[1].startTime = 30s
FogCloud.client[1].stopTime = 60s
FogCloud.client[1].coefficient = 2.0
FogCloud.client[0].taskAllocation = "softmax"
FogCloud.client[1].taskAllocation = "softmax"
FogCloud.client[0].routingProba = "0.7 0.2 0.1"
```

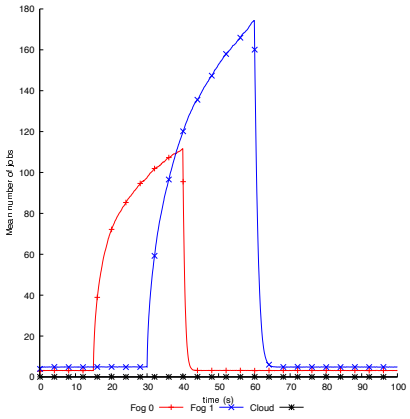
Strategy	fog[0]	fog[1]	fog[2]	Total
Static	24.7	43.5	0.0	68.2
ϵ-greedy	4.3	4.6	1.8	10.7
Sensible	4.5	4.6	0.9	10.0
Exp3	6.4	7.9	1.0	15.3
Softmax	5.0	5.1	0.9	11.0

Table 5.1: Mean number of jobs under the different task allocation strategies.

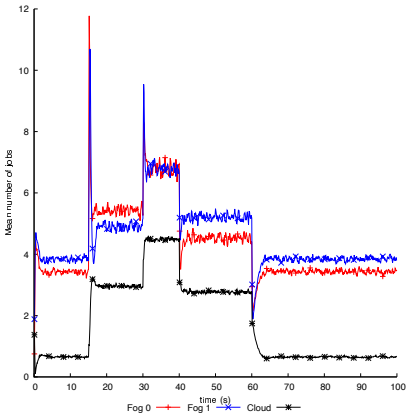
```
23 FogCloud.client[1].routingProba = "0.2 0.7 0.1"
24 repeat = 5000
```

The results obtained for this scenario are shown in Fig. 5.4, 5.5 and 5.6, as well as in Tables 5.1 and 5.2.

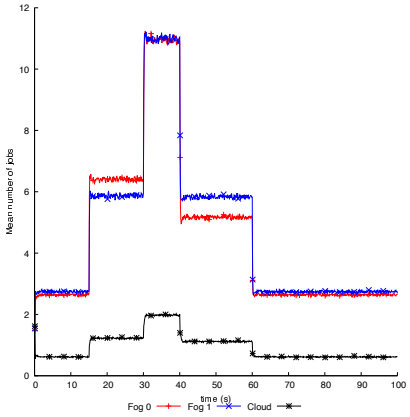
Let us first analyze Fig. 5.4, which presents the mean number of jobs as a function of time in each Fog node and in the Cloud under the different task allocation strategies. As expected, the static task allocation strategy in which jobs are routed to the nearest Fog node does not perform very well : between times t_0 and t_2 (resp. t_1 and t_3), the mean number of jobs at Fog node `fog[0]` (resp. `fog[1]`) grows at a very high pace to reach more than 100 jobs (resp. 170 jobs) in the system. Surprisingly, the EXP3 task allocation strategy is also not very efficient. In fact, this strategy adapts its routing probabilities to the variations of the input traffic, but the adaptation seems to be slower than that of the other algorithms. The softmax allocation performs quite well, but not as good as the sensible routing and $\epsilon - greedy$ strategies. Both strategies adapt very well and very fast to the variations of the input traffic. We note however that with the $\epsilon - greedy$ strategy, there are some undesirable peaks in the mean number of jobs, and also that this strategy sends more jobs to the Cloud than the sensible routing strategy. Table 5.1 gives the time-average number of jobs in each datacenter under each task allocation strategy. It is clear that, even though they are very simple and assume almost no knowledge of the infrastructure, the adaptive task allocation algorithms all lead to a significant reduction of the number of jobs in the system.



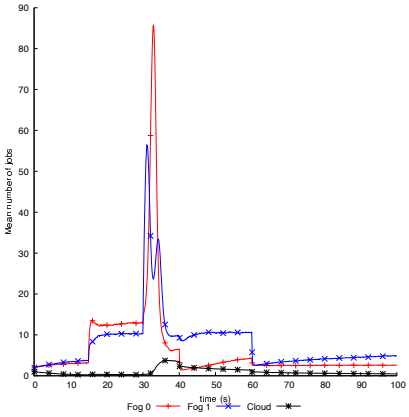
(a) Static task allocation



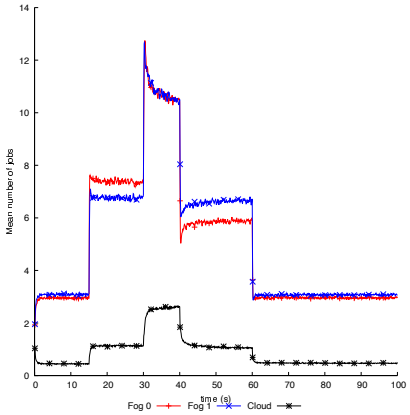
(b) ϵ-greedy algorithm



(c) Sensible routing



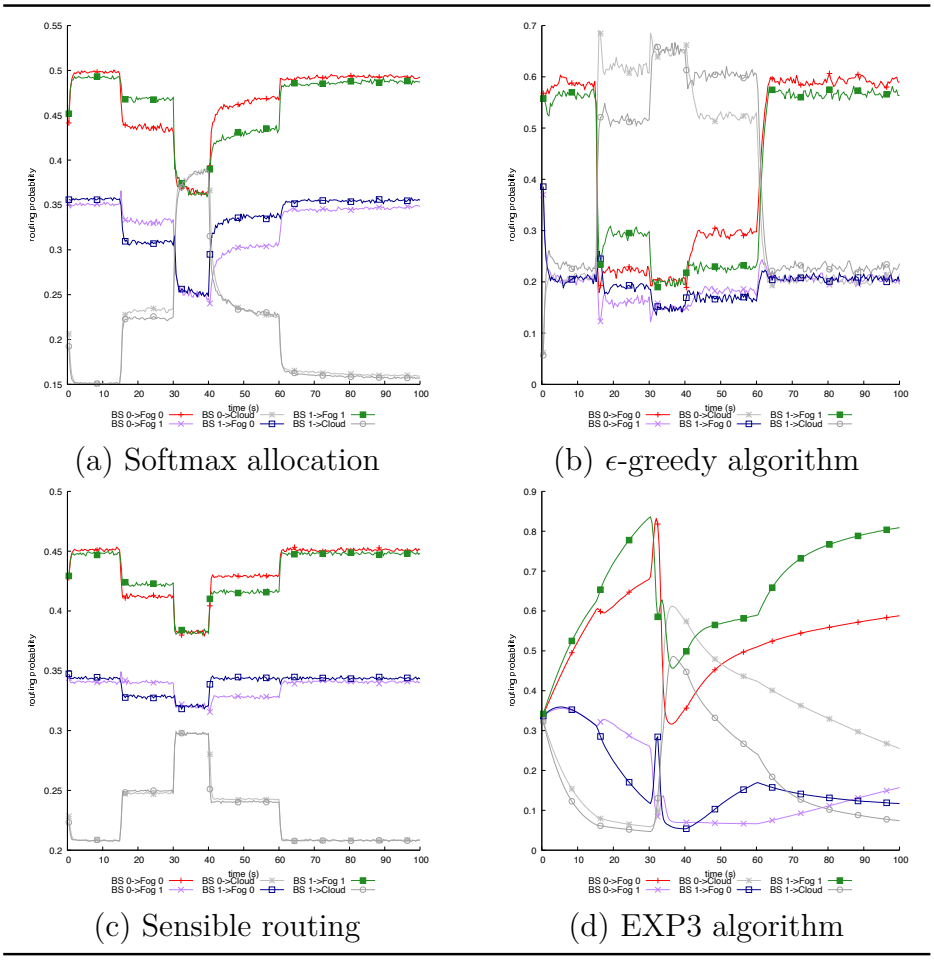
(d) EXP3 algorithm



(e) Softmax allocation

Figure 5.4: Mean number of jobs in each Fog node.

We now turn our attention to Figure 5.5, which shows the job routing probabilities as functions of time under the different task allocation strategies. Since these routing probabilities do not vary with the static task allocation strategy, this strategy is not included in Figure 5.5. We remark that the adaptation of routing probabilities is very slow for the EXP3 algorithm as compared to other dynamic strategies. We also note that under peak traffic conditions (between times t_1 and t_2), the ϵ-greedy strategy send much more jobs to the Cloud (almost 65% for both access nodes) than the sensible routing and softmax strategies, which sends only 30% and 40% of their jobs to the Cloud between times t_1 and t_2, respectively. The two latter strategies lead to similar routing probabilities.



(a) Softmax allocation

(b) ϵ-greedy algorithm

(c) Sensible routing

(d) EXP3 algorithm

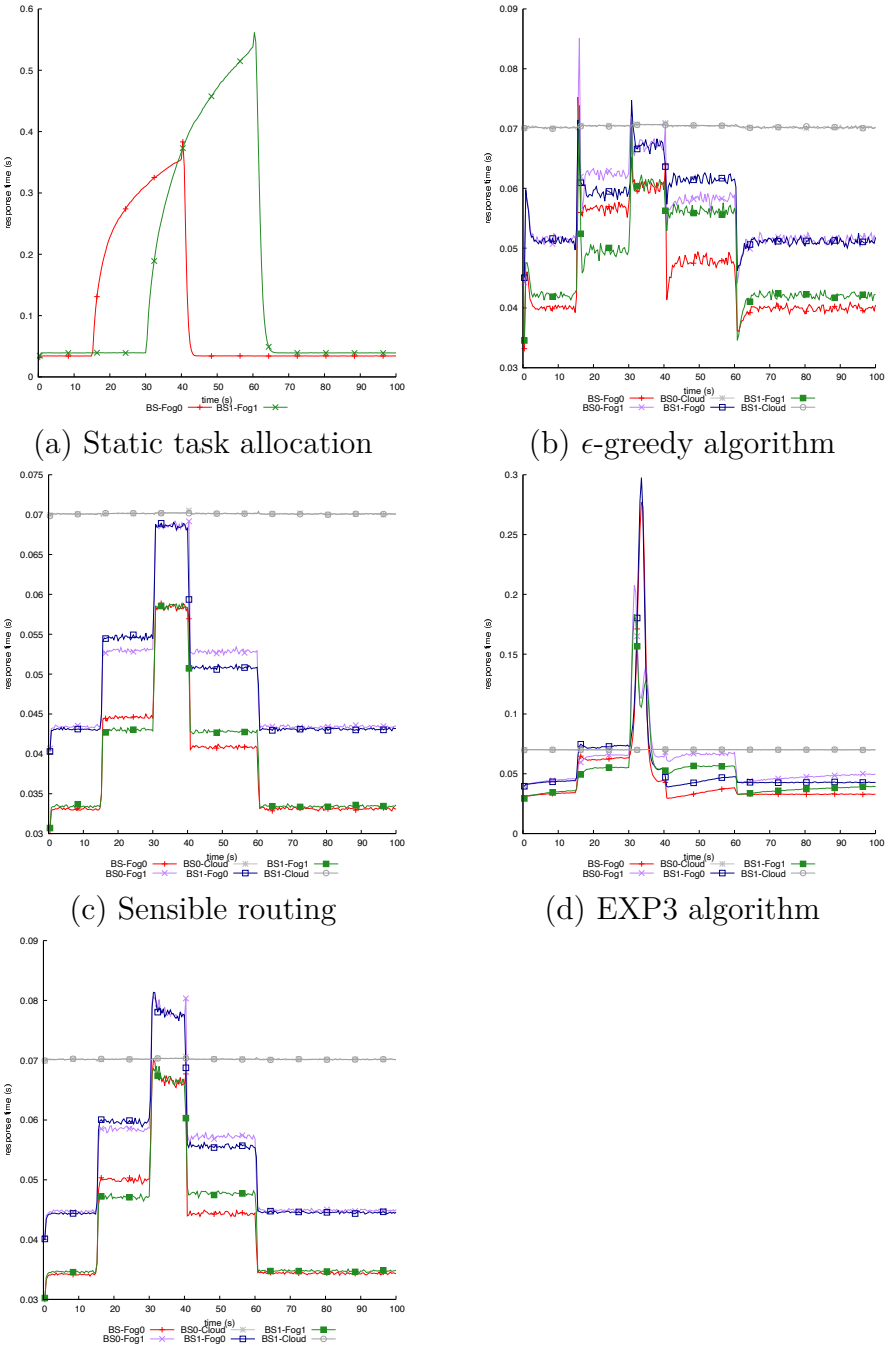
Figure 5.5: Task allocation strategies.

Finally, let us discuss the results presented in Figure 5.6, which shows the

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
Static	99.5	–	–	–	156.0	–
ϵ-greedy	46.1	56.1	70.3	56.1	48.0	70.3
Sensible	38.9	49.2	70.1	48.9	39.2	70.1
Exp3	45.2	59.2	70.1	54.7	49.0	70.1
Softmax	42.0	52.7	70.2	52.3	42.4	70.2

Table 5.2: Response times (ms) for each origin-destination pair under the different task allocation strategies.

jobs response times under the different task allocation strategies for each source-destination pairs as functions of time. As before, the static task allocation strategy has very bad performance, since its response times are an order of magnitude greater than that of dynamic strategies. The response times under the EXP3 algorithm are acceptable, except for a very high peak between 30 s and 36 s. As we already observed, this algorithm adapts to the brutal variations in the input traffic, but too slowly. The other algorithms perform quite well, in particular the sensible routing and ϵ-greedy algorithms. We note the remarkable performance of the sensible task allocation which allows to keep response times always below 70 ms. Table 5.2 gives the time-average values of the response time for every source-destination pair under the different task allocation strategies. It is clear from this table that dynamic task allocation strategies lead to a significant reduction of response times. We also note that the sensible task allocation strategy outperforms the other strategies.



(a) Static task allocation

(b) ϵ-greedy algorithm

(c) Sensible routing

(d) EXP3 algorithm

(e) Softmax allocation

Figure 5.6: Response times under the different task allocation strategies.

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
Static	100.2	-	-	-	156.1	-
ϵ-greedy	45.9	55.6	70.0	55.8	47.8	70.2
Sensible	38.9	49.1	70.1	48.8	39.2	70.2
Exp3	45.4	58.3	70.0	48.8	39.2	70.2
Softmax	41.9	52.5	70.1	52.3	42.4	70.2

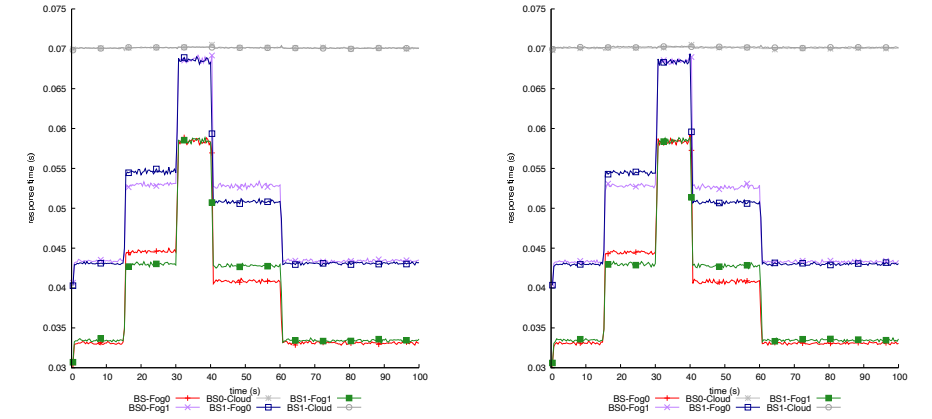
Table 5.3: Response times (ms) for each origin-destination pair under the different task allocation strategies under variable link delays.

B. Exponentially distributed Service Times - Variable Link Delays

In order to evaluate the impact of the variability of link delays on the performance, we present a configuration that takes this into consideration. Instead of constant delays we now use a uniform distribution to model them. More precisely, we replace the constant communication delay of 5 ms by a random delay with the same mean but uniformly drawn in the interval [3 ms, 7 ms] (the standard deviation is $\sigma = 1.15$). Similarly we replace fixed delays of 10 ms (resp 30 ms) by uniform random delays in the interval [4 ms, 16 ms] (resp. [20 ms, 40 ms]), which yields a standard deviation $\sigma = 3.46$ (resp. $\sigma = 5.77$).

Table 5.3 presents the response times under the different task allocation algorithms when link delays are variable. These values have to be compared to the mean response times obtained for fixed link delays, which are given in Table 5.2. Sweeping through all values in both tables, we see that the variation of the mean response times never exceeds 1.2%. We thus conclude that the adaptive algorithms considered in our simulations are relatively robust to a moderate variability in the communication delays.

Figure 5.7 compares the mean response times obtained with the sensible routing algorithm in both settings, that is, under fixed link delays and under variable link delays. We hardly notice the difference, which shows that this algorithm is robust to such perturbations.



(a) Fixed link delays. (b) Variable link delays

Figure 5.7: Response times obtained with the sensible task allocation algorithm under fixed link delays and under variable link delays.

C. Pareto-distributed Service Times - constant link delays

In order to evaluate the impact of the variability of job sizes on the task allocation algorithms, we now assume that the processing times of the job follow a Pareto distribution, instead of an exponential distribution. In other words, we assume that the probability that the processing time X of a job be greater than some number x is given by

$$\Pr(X > x) = \begin{cases} \left(\frac{x_{\mathrm{m}}}{x}\right)^{\alpha} & x \geq x_{\mathrm{m}}, \\ 1 & x < x_{\mathrm{m}}, \end{cases}$$

where x_{m} is the (necessarily positive) minimum possible value of the processing time, and α is a positive parameter. If $\alpha > 2$, the Pareto distribution has a finite mean and variance which are given by $\alpha x_{\mathrm{m}}/(\alpha - 1)$ and $\alpha x_{\mathrm{m}}^2/\left[(\alpha - 1)^2(\alpha - 2)\right]$. We consider two different Pareto distributions:

- **First Pareto distribution** - We choose $\alpha = 2.25$ and compute the minimum value x_{m} so as to keep the same mean values for the job processing times as in Section 5.5.1 (that is, 15 ms in Fog nodes and 10 ms in the Cloud). For the jobs executed in the Fog nodes (resp. in the Cloud), the standard deviation of the processing time is now 20 ms (resp. 13.3 ms) instead of 15 ms (resp. 10 ms) with the exponential distribution.
- **Second Pareto distribution** - We choose $\alpha = 2.05$ and, as before, we compute the minimum value x_{m} so as to keep the same mean values for the

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
Static	103.1	-	-	-	161.1	-
ϵ-greedy	50.5	59.8	70.7	59.6	52.9	70.7
Sensible	44.2	54.6	70.1	53.9	45.0	70.1
Exp3.	48.6	60.1	70.2	57.6	51.2	70.2
Softmax	52.3	61.2	70.5	65.1	55.2	70.5

Table 5.4: Response times (ms) for each origin-destination pair under the different task allocation strategies for the first Pareto distribution of job sizes.

job processing times. For the jobs executed in the Fog nodes (resp. in the Cloud), the standard deviation of the processing time is now 46.8 ms (resp. 31.2 ms) instead of 15 ms (resp. 10 ms) with the exponential distribution.

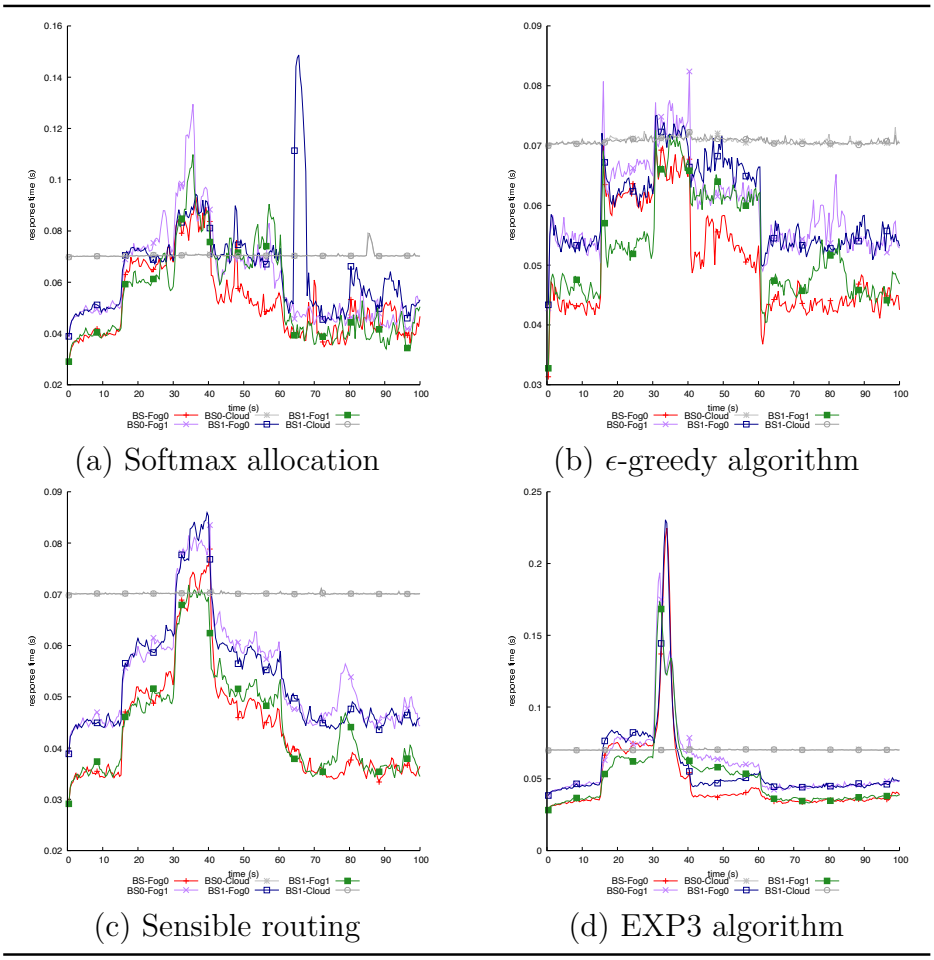
We note that the variability of job sizes is greater with the first Pareto distribution than with an exponential distribution, and that its is even greater with the second Pareto distribution.

Figure 5.8 shows the response times under the different job allocation schemes for the first Pareto distribution. Again, we observe that the EXP3 algorithm does not adapt fast enough to traffic variations. The best results are obtained with the ϵ-greedy and sensible allocations. The average response times obtained are reported in Table 5.4. We note that, although there are some response time peaks with this algorithm, the EXP3 allocation scheme provides better average results than the softmax allocation.

If we compare to the mean response times reported in Table 5.2, we see that all values increase when passing from exponentially-distributed service times to Pareto-distributed service times. This is something expected since it is well known that a greater variability in job sizes lead to larger response times. We note however the impact on response times is not the same for all task allocation algorithms. In order to assess the robustness of the algorithms with respect to the variability of job sizes, we consider the following metric for each access node i and each Fog node j

$$v_{i,j}^{A} = \frac{R_A^{Pareto}(i,j) - R_A^{Exp}(i,j)}{R_A^{Exp}(i,j)},$$

where $R_A^{Pareto}(i,j)$ (resp. $R_A^{Exp}(i,j)$) represents the mean response time of job



(a) Softmax allocation (b) ϵ-greedy algorithm

(c) Sensible routing (d) EXP3 algorithm

Figure 5.8: Response times under the different task allocation strategies for the first Pareto distribution of job sizes.

requests sent by access node i to Fog node j under task allocation algorithm A and for Pareto-distributed (resp. Exponentially-distributed) service times. Using the values in Tables 5.2 and 5.4, we can compute $v_{i,j}^{A}$ for each source-destination pair (i, j) and each algorithm A, and compare the increase in response times obtained under the different algorithms. This comparison is done in Figure 5.9 where the minimal, maximal and average increases relative to the exponential distribution are displayed for each algorithm. Interestingly, the EXP3 allocation scheme seems to be more robust to the variability of job sizes. Indeed, we see that with EXP3 the mean response times increase by at most 7.5% when we pass from an exponential distribution to the first Pareto distribution. The increase

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
Static	127.1	-	-	-	192.4	-
ϵ-greedy	59.3	66.0	71.4	67.9	59.9	71.5
Sensible	57.9	69.5	70.2	67.4	60.0	70.2
Exp3.	59.1	64.2	70.3	66.0	57.9	70.4
Softmax	54.1	70.9	70.7	73.4	72.3	70.7

Table 5.5: Response times (ms) for each origin-destination pair under the different task allocation strategies for the second Pareto distribution of job sizes.

in response times is as high as 30.2% (resp. 14.8%) for the Softmax allocation (resp. sensible allocation). Although slightly less robust, the ϵ-greedy allocation is also quite robust to the variability of job sizes since the increase in the mean response times is at most 10.2%. While the minimal variations observed are non-significant, we see that the average relative increases is smallest in EXP3 with 3.2%, confirming its good performance. ϵ-greedy remains quite robust when looking at that metric with an increase of 5.7%, followed by sensible routing at 8.6%. However, Softmax allocation does not handle the variation well with an average increase of 16%.

Similarly Table 5.5 and Figure 5.10 present the results obtained with the second Pareto distribution, comparing them with the exponential distribution. The response times from the exponential distribution to the second Pareto distribution increase the least in ϵ-greedy and EXP3 allocation with $28,6\%$ and $30,7\%$ increase. We note much larger gaps in Sensible routing and Softmax with 53% (resp. 70%). The average increase is best in EXP3 with 13.1%, then with ϵ-greedy with 16%. Surprisingly, while the maximal relative increase was significantly worse in Softmax, Sensible routing and Softmax allocation both obtain comparable average increases with respectively 30.3% and 29.3%.

5.5.2 Task Allocation with offloading

The second scenario is similar to the first one, except that the Fog nodes are now connected to the Cloud (Figure 5.11). The communication delay between the Fog nodes and the Cloud is constant and equal to 28 ms. Note that this value was chosen so as to satisfy the triangle inequality. In this scenario, Fog nodes can offload jobs to the Cloud. All other parameters are similar to those used in

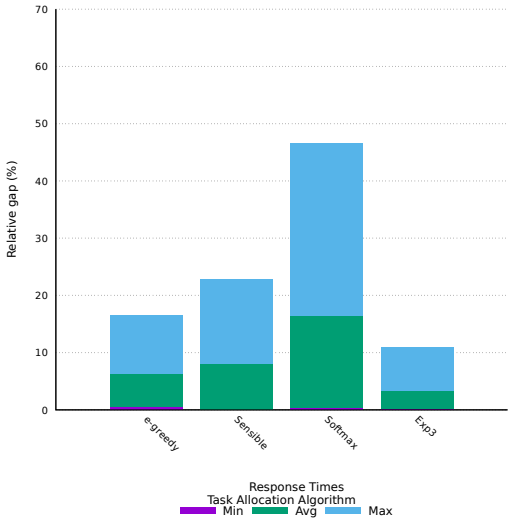


Figure 5.9: Relative performance degradation between the first Pareto distribution and exponential distribution for service times across task allocation algorithms

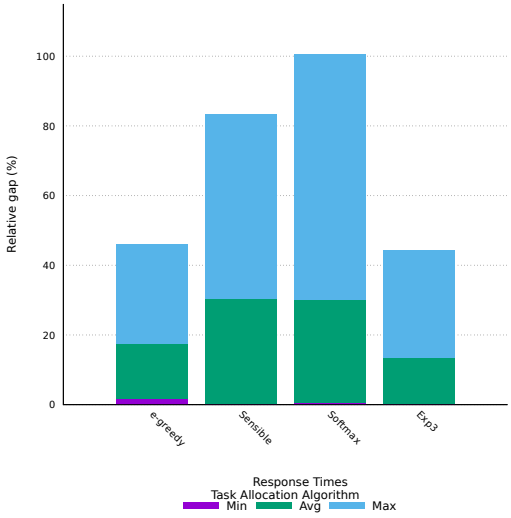


Figure 5.10: Relative performance degradation between the second Pareto distribution and exponential distribution for service times across task allocation algorithms

Scenario 1.

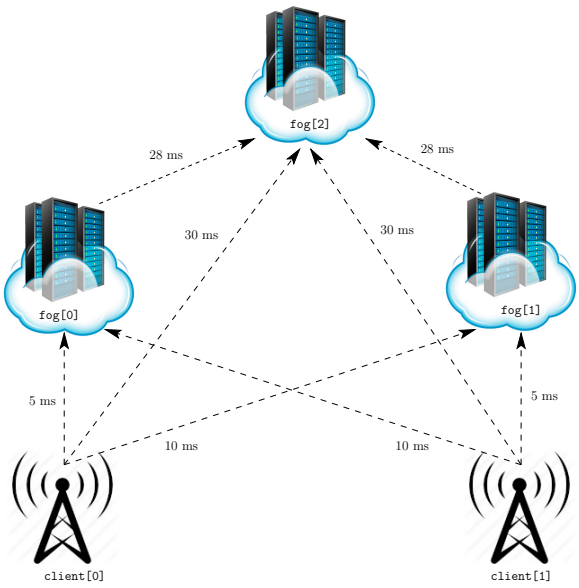
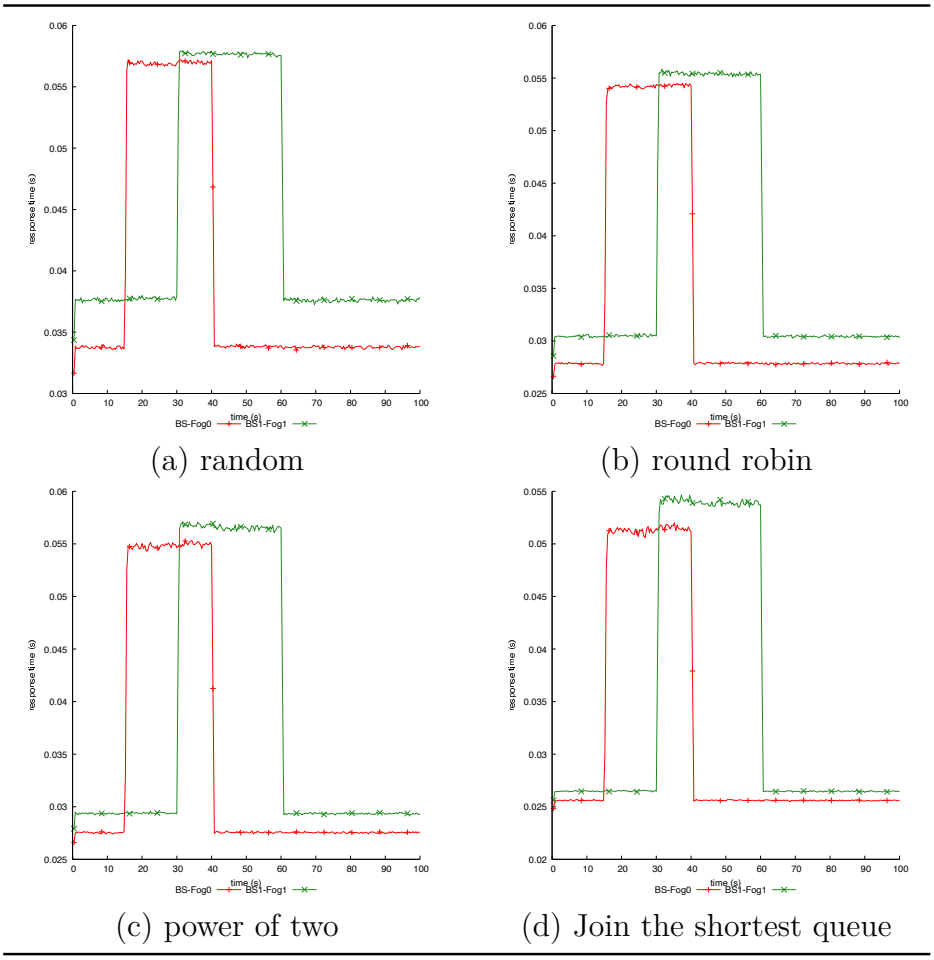


Figure 5.11: A scenario in which Fog nodes can offload jobs to the Cloud.

The offloading mechanism that we consider assumes that the execution time of a job in the Cloud is constant and equal to the processing time of the job in the Cloud (i.e., it assumes that a job request sent to the Cloud always find a free server). Under this assumption, a job request offloaded to the Cloud c by Fog node j will have a response time equal to $\ell_{j,c}+\frac{1}{\mu_c}+\ell_{c,j}$, which yields 66 ms with the values used in this scenario. The mechanism is then as follows. Upon reception of a job, the dispatcher of the Fog node j first selects the application server n for executing this job. It then queries the number of jobs q_n at this server, so as to estimate the execution time of the job using the formula $(q_n+1)\times\frac{1}{\mu_j} = (q_n+1)\times 15$ ms. If this execution time is greater than the response time obtained by offloading the job to the Cloud, then the job is offloaded to the Cloud.

We note that this dispatching mechanism requires the dispatcher to keep track of the number of jobs executing at each server. We also note that this mechanism depends on the job dispatching scheme used in the Fog nodes. If the Fog nodes use the "Join the Shortest Queue" scheme, then the execution time estimated by the dispatcher corresponds to the minimum execution time that can be achieved. However, for another dispatching scheme such as a random allocation, the estimated execution time will be greater. We emphasize that the information used in this offloading mechanism is completely different from the

one used by the adaptive routing algorithms discussed so far, even though the information used by the offloading mechanism is either local (number of jobs executing at each server) or static (response time from the Cloud). Let us first focus on the offloading mechanism and compare the response times obtained with different job dispatching mechanism. We assume that jobs are always sent to the nearby micro-datacenter, where they can be offloaded to the Cloud. Figure 5.12 presents the response times obtained with four dispatching schemes: random, round robin, power of two and join the shortest queue.



(a) random (b) round robin

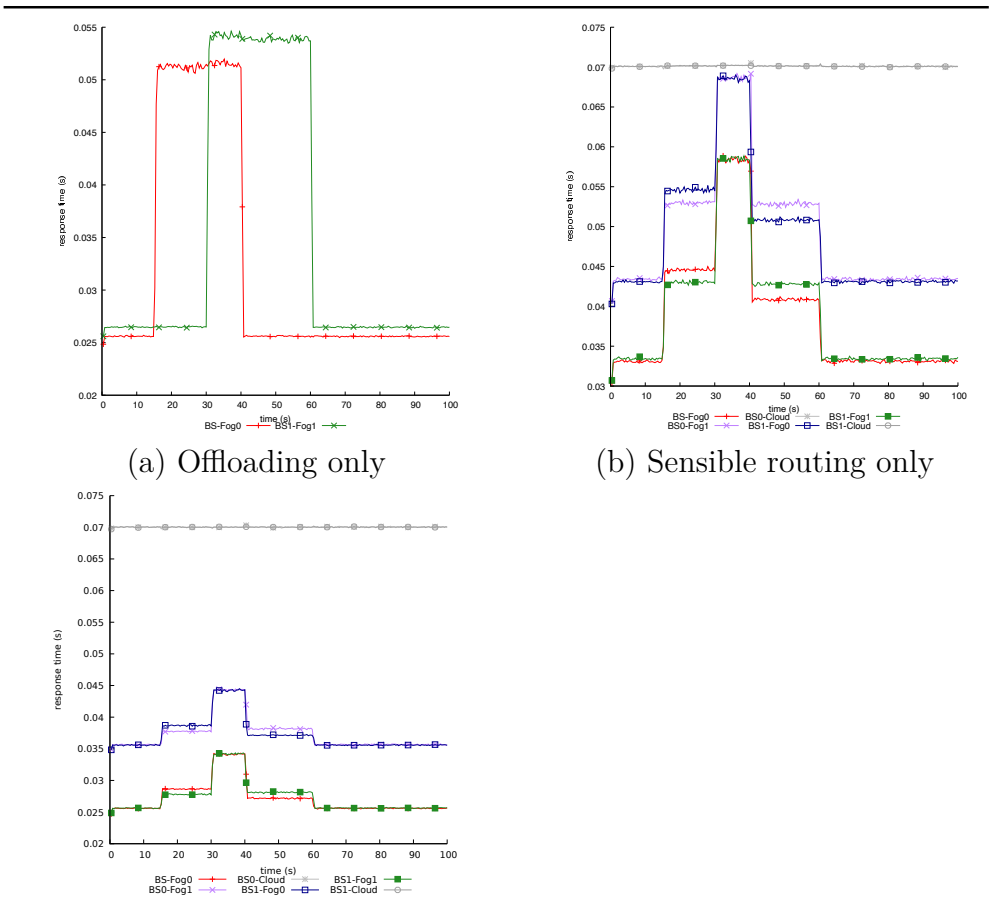
(c) power of two (d) Join the shortest queue

Figure 5.12: Response times with task offloading under the different dispatching policies.

As can be seen, the round robin and power-of-two policies provide significantly better results than the random policy. The Join-The-Shortest-Queue however

clearly outperforms all other policies. In the following, we shall assume that this dispatching strategy is used in all micro-datacenters as well as in the Cloud.

Figure 5.13 compares the performance obtained under different strategies. The first one uses only job offloading to the Cloud by Fog nodes and a static allocation strategy routing tasks to the closest node (in this case always Fog node 0 for the first base station and Fog node 1 for the second base station). The second one uses only an adaptive task allocation strategy (which is sensible routing in this case), but once affected to a Fog node, jobs cannot be offloaded to the Cloud. Finally, the third one corresponds to the case where both mechanisms are combined.



(a) Offloading only

(b) Sensible routing only

(c) Offloading and sensible routing

Figure 5.13: Sensible routing vs Offloading only vs Sensible routing and Offloading for exponentially distributed service times

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
ϵ-greedy	33.7	44.7	70.0	44.7	34.7	70.0
Sensible	27.2	37.3	70.0	37.2	27.3	70.0
Exp3.	29.2	41.7	69.9	39.2	31.7	70.0
Softmax	27.9	38.2	70.0	38.0	28.1	70.0

Table 5.6: Response times (ms) obtained in the second scenario by combining job offloading and adaptive task allocation in the case of exponentially-distributed service times.

	client[0]			client[1]		
Strategy	fog[0]	fog[1]	fog[2]	fog[0]	fog[1]	fog[2]
ϵ-greedy	34.6	45.4	70.1	45.6	35.7	70.1
Sensible	27.0	37.1	70.0	37.0	27.1	70.0
Exp3.	28.8	41.1	69.9	38.8	31.0	69.9
Softmax	28.2	38.4	70.0	38.2	28.4	70.0

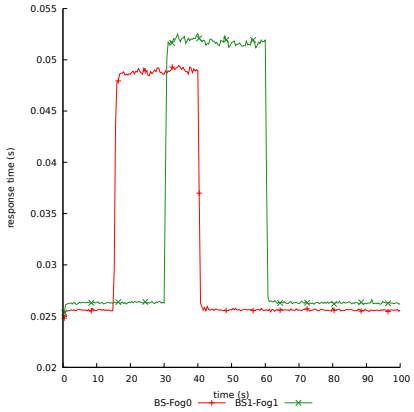
Table 5.7: Response times (ms) obtained in the second scenario by combining job offloading and adaptive task allocation for the first Pareto distribution of service times.

As we can observe the offloading mechanism enables a significant reduction of response times. This mechanism outperforms adaptive task allocation strategies in this scenario. We note however that the use of an adaptive task allocation algorithm in combination with an offloading strategy drops response times.

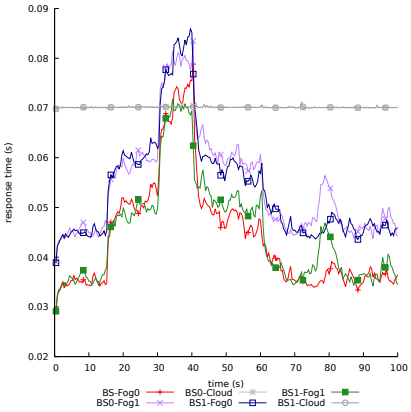
Pareto distributed Service Times

Once again, we aim to evaluate the impact of the variability of job sizes on the task allocation algorithms. We compare the results obtained with the offloading configuration using an exponential service distribution and the first Pareto distribution described in subsection 5.5.1.

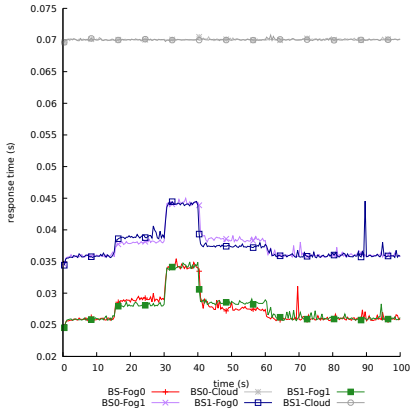
We compare the three strategies previously mentioned in Figure 5.14 but this time using the first Pareto distribution for service times.



(a) Offloading only



(b) Sensible routing only



(c) Offloading and sensible routing

Figure 5.14: Sensible routing vs Offloading only vs Sensible routing and Offloading for the first Pareto distribution for service times

As a reminder the first one uses only job offloading to the Cloud by Fog nodes with a JSQ dispatching scheme and a static allocation strategy. The second one uses only the sensible routing adaptive algorithm and in the third one we have a combination of both strategies. We notice that the combined strategy resists well to the variability induced by the Pareto distribution, as the shape and the scale of values remain comparable to the exponential distribution. However, we can see some jitter but reasonably small. We conclude that this strategy is robust in the face of job size variability.

We look at these results further in Table 5.7 showing the average response times from our simulation using the first Pareto distribution. Comparing it with Table 5.6 which uses the exponentially distribution for service times, we notice

surprisingly that EXP3 and sensible routing perform slightly better in this distribution with an average decrease of response times of 1% (resp. 0.4%). The Softmax and ϵ-greedy algorithms, on the other hand, perform worse with average response times seeing a maximum increase of 1.1% (and 2.9%). We suppose that the relative stability of these algorithms under job size variability is compensated by the offloading strategy in the Cloud which always can handle the jobs even when the Fog node queues are full due to the increase in the service times.

5.6 Summary and Discussion

In summary, we have proposed a simulation model of a Fog Computing infrastructure which relies on the network simulation framework OMNET++

We have simulated two configurations: one where two base stations are connected to two Fog nodes and one Cloud node. The other is similar but with extra connection from Fog nodes to the Cloud node to give Fog nodes the option to offload jobs to the Cloud.

Each base station has an allocation strategy from static routing (send to the closest Fog node) to adaptive allocations such as ϵ-greedy, Softmax, EXP3 and sensible routing. We compare the performance among these strategies against the two configurations and a number of parameters such as link delay times or service processing times distributions. We also include in all scenarios traffic perturbations to evaluate how well the algorithms react.

We first observe that any adaptive algorithm, even though simple, outperforms the static routing strategy in terms of average response times.

In the first configuration we have noticed that sensible routing provides the best response times. We have compared the robustness of the different task allocation schemes with respect to variations in the processing times. In general, we have found Exp3 to be more robust than other task allocation strategies.

In the second configuration, we examine response times across the algorithms in combination with various dispatching strategies. The strategy used for offloading relies on the state of the Fog nodes and on its expectation of the processing time by the Cloud. Four local dispatching schemes are used: *random*, *round-robin*, *join-the-shortest-queue* and *power-of-two choices*. We observe that *join-the-shortest-queue* outperforms all other policies. We also notice significant gains on response times when Fog nodes are allowed to offload their tasks to the Cloud. The reduction in response times is even more significant when an offloading mechanism is combined with an adaptive task allocation strategy. When comparing the performance for highly variable job sizes, we note that the offloading strategy mitigates the variability of job sizes and that the EXP3 and the sensible routing

algorithms are barely affected.

Chapter 6

Conclusion and Future Work

Fog computing promises modern service infrastructures that guarantee low latency, fault-tolerance and a seamless coordination with the Cloud. In this vision, IoT devices, from sensors to vehicles and smart buildings would rely on locally distributed computing and storage resources. However, the technology is still in its infancy and there are numerous challenges to address to make it a reality. In this thesis, we have studied three major problems related to the design and operation phases of a Fog network:

- How should Fog networks be built and capacitated based on traffic expectations in order to minimize the overall costs while providing the expected quality of service?
- How should Fog controllers migrate services to minimize the associated overheads in the context of user mobility?
- How should base stations allocate jobs to Fog nodes given a fully decentralized architecture where base stations compete for resources?

In Chapter 3, we have proposed a MILP formulation of the optimization problem. We have implemented it using the commercial solver Gurobi and compared the performance of centralized and decentralized solutions. This work was based on the assumption of exponentially distributed service times, Bernoulli routing and the use of simple queuing theoretical models. A possible extension is to consider more elaborated queuing models using general service time distributions and look at analytical approximations that can help dimension the system. Another improvement that would make the problem statement more realistic would be to consider the capacity of individual servers, priority mechanisms for different classes of jobs, or more advanced resource sharing mechanisms. Finally, it would

be interesting to evaluate other load-balancing policies such as the Power of Two Choices or Join the Shortest Queue First.

In Chapter 4, we have designed a model to help minimize service migrations from an handover traffic matrix and an heuristic based on a weighted set cover approximation. As a future work, we have suggested a path-based approach that requires to track user mobility patterns but would provide better results. Another interesting challenge would be to use a finer-grained model for services. In practice Cloud and Fog applications are composed of numerous microservices (including database services), which depend on each other with different latency requirements. It would be valuable to study the optimal assignment of these different microservices on Fog nodes while considering mobility patterns. Because we expect the changes to be incremental as users move, this might be solved as an online optimization problem. However the many variables involved might also make it suitable to learning-based algorithms.

In Chapter 5, we have proposed a Fog simulation model based on the OMNeT++ and used it to compare the performance of various task allocation algorithms. We have studied their reaction to link delay and job size variability, and the benefits of offloading jobs. Our experiments have shown that among the algorithms evaluated, Sensible routing performs well in the general case and EXP3 is particularly robust to variability. Giving Fog nodes the opportunity to offload jobs seems to always provide better response times. An observation is that since each base station optimizes independently the response time of its own jobs, the base stations are involved in a non-cooperative routing game. As a consequence, if a distributed task allocation algorithm converges to an equilibrium allocation, then this allocation is a Nash equilibrium of the game. We note that in general, a Nash equilibrium does not provide any guarantee on its global performance and in the worst case, its performance can be arbitrarily far from an optimal configuration. In future works, it would be interesting to implement a distributed model-based task allocation and compare with the decentralized solution and its Nash equilibrium in order to study the Price-of-Anarchy. Another interesting follow-up would be to run the simulation with many more base station and Fog node instances to compare our results and see how these algorithms react under scale. It would also be interesting to run the task allocation experiments on more realistic environments such as the Mininet emulator or on real-world servers implementing the strategies, to compare the simulated results and ensure they conform to real-life experiments. It would be useful to compare more state-of-the-art multi-armed bandits algorithms such as Thompson sampling and UCB, or more elaborate reinforcement learning algorithms, and extend the scenario matrix to analyze in more depth the strengths of each algorithms. For instance

certain algorithms may be able to learn and predict periodic perturbations and minimize their impact on the users. Finally, it might be interesting to extend the usage of the simulation model we have proposed to support other scenarios, for instance to model service migrations and their performance.

All these techniques we have proposed can be used in conjunction to help designing Fog infrastructures. We have found the key challenges of Fog computing to relate mostly to:

- The vision of Fog computing being essentially a trade-off between centralization and decentralization, where we aim to decentralize *as much as required* and *as little as we can* to both keep the proposed SLAs and enable economies of scale while keeping the energy requirements as low as possible;
- Mobility being the norm in the Fog, where proposing transparent handovers and keeping low-latency is a basic expectation of Fog consumers which needs to be explicitly considered;
- The Fog being an application platform rather than merely a network infrastructure such as 5G and Mobile networks. This requires considerations of supporting modern applications that work using complex protocols and have data storage requirements that fall under the Fog's SLA;
- User privacy, though it is not directly addressed in this thesis, is an important consideration, for instance location privacy with respect to tracking could be a limitation in the path based mobility approach we have proposed in Chapter 4 for service migration minimization;
- Fault-tolerance, which is not addressed either here, remains an important concern. It is a benefit of distributed architecture but it needs to be considered using replication.

These challenges have been guiding the research in this thesis and made our exploration stimulating, but there is much left to do for the research community! Thanks for reading!

Bibliography

[Aarag 2001] H. El Aarag and M. Bassiouni. *Performance evaluation of TCP connections in ideal and non-ideal network environments*. Computer Communications, vol. 24, pages 1769–1779, 2001.

[Adan 2001] I. Adan and J. Resing. Queueing theory: Ivo adan and jacques resing. Eindhoven University of Technology. Department of Mathematics and Computing Science, 2001.

[AEC] *Automotive Edge Computing Consortium*. https://aecc.org/.

[Agarwal 2010] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman and Harbinder Bhogan. *Volley: Automated Data Placement for Geo-distributed Cloud Services*. In Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, pages 2–2, Berkeley, CA, USA, 2010. USENIX Association.

[Aissioui 2018] A. Aissioui, A. Ksentini, A. M. Gueroui and T. Taleb. *On Enabling 5G Automotive Systems Using Follow Me Edge-Cloud Concept*. IEEE Transactions on Vehicular Technology, vol. 67, no. 6, pages 5302–5316, June 2018.

[A.S Tanenbaum 2006] Maarten Van Steen A.S Tanenbaum. Distributed systems: Principles and paradigms (2nd edition). Prentice Hall, 2 edition, 2006.

[Auer 2002] P. Auer, N. Cesa-Bianchi, Y. Freund and R. E. Schapire. *The nonstochastic multiarmed bandit problem*. SIAM Journal on Computing, vol. 32, no. 1, pages 48–77, 2002.

[Baek 2019] Jung-Yeon Baek, G. Kaddoum, S. Garg, K. Kaur and V. Gravel. *Managing Fog Networks using Reinforcement Learning Based Load Balancing Algorithm*. 2019 IEEE Wireless Communications and Networking Conference (WCNC), pages 1–7, 2019.

[Beraldi 2017] R. Beraldi, A. Mtibaa and H. Alnuweiri. *Cooperative load balancing scheme for edge computing resources.* 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), pages 94–100, 2017.

[Bertsimas 1997] D. Bertsimas and J.N. Tsitsiklis. Introduction to linear optimization. Athena Scientific, 1997.

[Bittencourt 2015] L. F. Bittencourt, M. M. Lopes, I. Petri and O. F. Rana. *Towards Virtual Machine Migration in Fog Computing.* In 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), pages 1–8, Nov 2015.

[Bonomi 2011] F. Bonomi. *Cloud and Fog Computing: Trade-offs and applications.* In International Symposium of Computer Architecture, 2011.

[Bonomi 2012] F. Bonomi, R. Milito, J. Zhu and S. Addepalli. *Fog Computing and Its Role in the Internet of Things.* In ACM SIGCOMM International Conference on Mobile Cloud Computing, 2012.

[Boyd 2004] Stephen Boyd and Lieven Vandenberghe. Convex optimization. Cambridge University Press, 2004.

[Brown 2007] Gerald Brown and Robert Dell. *Formulating Integer Linear Programs: A Rogues' Gallery.* Informs Transactions on Education, vol. 7, pages 153–159, 01 2007.

[Ceselli 2015] Alberto Ceselli, Marco Premoli and Stefano Secci. *Cloudlet network design optimization.* In Rahim Kacimi and Zoubir Mammeri, editors, Networking, pages 1–9. IEEE, 2015.

[Cheng 2019] Xiaolan Cheng, Xiaoxia Zhou, Congfeng Jiang and Jian Wan. *Toward Computation Offloading in Edge Computing: A Survey.* 2019.

[CIS] *CISCO Fog Computing White paper.* https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf.

[Codecá 2017] Lara Codecá, Raphaël Frank, Sébastien Faye and Thomas Engel. *Luxembourg SUMO Traffic (LuST) Scenario: Traffic Demand Evaluation.* IEEE Intelligent Transportation Systems Magazine, vol. 9, no. 2, pages 52–63, 2017.

[Coucheney 2015] P. Coucheney, B. Gaujal and P. Mertikopoulos. *Penalty-Regulated Dynamics and Robust Learning Procedures in Games*. Mathematics of Operations Research, INFORMS, vol. 40, no. 3, pages 611–633, 2015.

[Cuervo 2010] Eduardo Cuervo, Aruna Balasubramanian, Dae ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra and Paramvir Bahl. *MAUI: making smartphones last longer with code offload.* pages 49–62. ACM, 2010.

[D'Ambrosio 2010] C. D'Ambrosio, A. Lodi and S. Martello. *Piecewise linear approximation of functions of two variables in MILP models*. Operations Research Letters, vol. 38, pages 39–46, 2010.

[Dolui 2017] K. Dolui and S. K. Datta. *Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing*. In 2017 Global Internet of Things Summit (GIoTS), pages 1–6, 2017.

[ets 2009] *ETSI TR 102 638:Vehicular Communications; Basic Set of Applications; Definitions*. Technical report, ETSI Std. ETSI ITS Specification TR 102 638 version 1.1.1, June 2009.

[Fan 2017] Qiang Fan and Nirwan Ansari. *Cost Aware cloudlet Placement for big data processing at the edge*. In ICC, pages 1–6. IEEE, 2017.

[Fas] *Fastly Edge Computing*. https://vimeo.com/267273182.

[Flinn 2009] J. Flinn and Ya-Yunn Su. *Replication-based cyber foraging and automated configuration management*. 2009.

[Fricker 2016] C. Fricker, F. Guillemin, P. Robert and Guilherme Thompson. *Analysis of an Offloading Scheme for Data Centers in the Framework of Fog Computing*. ArXiv, vol. abs/1507.05746, 2016.

[Gelenbe 1989] E. Gelenbe. *Random Neural Networks with Negative and Positive Signals and Product Form Solution*. Neural Computation, vol. 1, pages 502–510, 1989.

[Gelenbe 2003] E. Gelenbe. *Sensible decisions based on QoS*. Computational Management Science, vol. 1, pages 1–14, 2003.

[Gelenbe 2012] Erol Gelenbe, Ricardo Lent and Markos Douratsos. *Choosing a Local or Remote Cloud*. In Second Symposium on Network Cloud Computing and Applications, NCCA 2012, London, United Kingdom, December 3-4, 2012, pages 25–30, 2012.

[Gonçalves 2018] Diogo Gonçalves, Karima Velasquez, Marília Curado, Luiz Fernando Bittencourt and Edmundo Roberto Mauro Madeira. *Proactive Virtual Machine Migration in Fog Environments*. 2018 IEEE Symposium on Computers and Communications (ISCC), pages 00742–00745, 2018.

[Gupta 2016] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh and Rajkumar Buyya. *iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments*. Softw., Pract. Exper., vol. 47, pages 1275–1296, 2016.

[Gurobi] Gurobi. *Gurobi Solver, url = https://www.gurobi.com/.*

[Gurobi Optimization 2018] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*, 2018.

[Harchol-Balter 2013] Mor Harchol-Balter. Performance modeling and design of computer systems : queueing theory in action. Cambridge University Press, 2013.

[Jia 2017] M. Jia, J. Cao and W. Liang. *Optimal Cloudlet Placement and User to Cloudlet Allocation in Wireless Metropolitan Area Networks*. IEEE Transactions on Cloud Computing, vol. 5, no. 4, pages 725–737, Oct.-Dec. 2017.

[Jonglez 2016] B. Jonglez and B. Gaujal. *Distributed Adaptive Routing in Communication Networks*. Research Report RR-8959, Inria ; Univ. Grenoble Alpes, October 2016.

[Jonglez 2017] B. Jonglez and B. Gaujal. *Distributed and Adaptive Routing Based on Game Theory*. In ALGOTEL 2017 - 19èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications, Quiberon, France, May 2017. hal-01517911.

[Kiani 2018] Abbas Kiani, Nirwan Ansari and Abdallah Khreishah. *Hierarchical Capacity Provisioning for Fog Computing*. CoRR, vol. abs/1807.01093, 2018.

[Kleinrock 1975] Leonard Kleinrock. Theory, volume 1, queueing systems. Wiley-Interscience, New York, NY, USA, 1975.

[Lattimore 2020] Tor Lattimore and Csaba Szepesvári. Bandit algorithms. Cambridge University Press, 2020.

[Levitin 2002] Anany V. Levitin. Introduction to the design and analysis of algorithms. Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[Li 2019] J. Li, X. Shen, L. Chen, D. P. Van, J. Ou, L. Wosinska and J. Chen. *Service Migration in Fog Computing Enabled Cellular Networks to Support Real-Time Vehicular Communications*. IEEE Access, vol. 7, pages 13704–13714, 2019.

[Lin 2019] L. Lin, Xiaofei Liao, H. Jin and Peng Li. *Computation Offloading towards Edge Computing*. 2019.

[Liu 2018] Liqing Liu, Zheng Chang, Xijuan Guo, S. Mao and T. Ristaniemi. *Multiobjective Optimization for Computation Offloading in Fog Computing*. IEEE Internet of Things Journal, vol. 5, pages 283–294, 2018.

[Lopes 2017] Márcio Moraes Lopes, Wilson A. Higashino, Miriam A.M. Capretz and Luiz Fernando Bittencourt. *MyiFogSim: A Simulator for Virtual Machine Migration in Fog Computing*. In Companion Proceedings of the10th International Conference on Utility and Cloud Computing, UCC '17 Companion, pages 47–52, New York, NY, USA, 2017. ACM.

[Marzie Zarinbal (auth.) 2009] Masoud Hekmatfar (eds.) Marzie Zarinbal (auth.) Reza Zanjirani Farahani. Facility location: Concepts, models, algorithms and case studies. Contributions to Management Science. Physica-Verlag Heidelberg, 1 edition, 2009.

[MEC] *Multi-Access Edge Computing*. https://www.etsi.org/technologies/multi-access-edge-computing.

[Mehta 2016] Amardeep Mehta, William Tärneberg, Cristian Klein, Johan Tordsson, Maria Kihl and Erik Elmroth. *How beneficial are intermediate layer Data Centers in Mobile Edge Networks?* In Workshops on Fog and Mobile Edge Computing at Foundations and Applications of Self* Systems, pages 222–229. IEEE–Institute of Electrical and Electronics Engineers Inc., 2016.

[Michele Conforti 2014] Giacomo Zambelli (auth.) Michele Conforti Gérard Cornuéjols. Integer programming. Graduate Texts in Mathematics 271. Springer International Publishing, 1 edition, 2014.

[Mims 2014] C. Mims. *Forget 'the cloud': 'the Fog' is Tech's Future*. The Wall Street Journal, 2014.

[Mondal 2017] Sourav Mondal, Goutam Das and Elaine Wong. *A Novel Cost Optimization Framework for Multi-Cloudlet Environment over Optical Access Networks*. GLOBECOM 2017 - 2017 IEEE Global Communications Conference, pages 1–7, 2017.

[NIS] *Fog Computing Conceptual Model*. https://www.nist.gov/publications/fog-computing-conceptual-model.

[NTT 2014] *Announcing the 'Edge Computing' concept and the 'Edge accelerated Web platform' prototype to improve response time of cloud*. NTT Press release, 2014.

[OPE] *OpenFog Reference Architecture*. http://site.ieee.org/denver-com/files/2017/06/OpenFog_Reference_Architecture_2_09_17-FINAL-1.pdf.

[Ouyang 2018] T. Ouyang, Z. Zhou and X. Chen. *Follow Me at the Edge: Mobility-Aware Dynamic Service Placement for Mobile Edge Computing*. IEEE Journal on Selected Areas in Communications, vol. 36, no. 10, pages 2333–2345, Oct 2018.

[Pathan 2012] Khan Pathan and Rajkumar Buyya. *A Taxonomy and Survey of Content Delivery Networks*. 04 2012.

[Pham 2019] Quoc-Viet Pham, Fang Fang, Vu Nguyen Ha, Md. Jalil Piran, Mai Le, Long Bao Le, Won-Joo Hwang and Zhiguo Ding. *A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art*, 2019.

[Pióro 2004] Michal Pióro and Deepankar Medhi. Routing, flow, and capacity design in communication and computer networks. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[Porambage 2018] Pawani Porambage, Jude Okwuibe, Madhusanka Liyanage, Tarik Taleb and Mika Ylianttila. *Survey on Multi-Access Edge Computing for Internet of Things Realization*. IEEE Communications Surveys & Tutorials, vol. PP, 06 2018.

[Qayyum 2018] T. Qayyum, A. W. Malik, M. A. Khan Khattak, O. Khalid and S. U. Khan. *FogNetSim++: A Toolkit for Modeling and Simulation of Distributed Fog Environment*. IEEE Access, vol. 6, pages 63570–63583, 2018.

[Rejiba 2019] Zeineb Rejiba, Xavier Masip-Bruin and Eva Marín-Tordera. *A Survey on Mobility-Induced Service Migration in the Fog, Edge, and Related Computing Paradigms*. ACM Comput. Surv., vol. 52, no. 5, pages 90:1–90:33, September 2019.

[Strang 2005] Gilbert Strang. Linear algebra and its applications (4ed). Brooks Cole, 4th edition, 2005.

[Sun 2017] Xiang Sun and Nirwan Ansari. *Latency Aware Workload Offloading in the Cloudlet Network*. IEEE Communications Letters, vol. 21, pages 1481–1484, 2017.

[Sutton 2018] R. S. Sutton and A. G. Barto. Reinforcement learning, an introduction (second edition). MIT Press, 2018.

[Taleb 2013a] T. Taleb, P. Hasselmeyer and F. G. Mir. *Follow-Me Cloud: An OpenFlow-Based Implementation*. In 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, pages 240–245, Aug 2013.

[Taleb 2013b] T. Taleb and A. Ksentini. *An analytical model for Follow Me Cloud*. In 2013 IEEE Global Communications Conference (GLOBECOM), pages 1291–1296, Dec 2013.

[Taleb 2013c] T. Taleb and A. Ksentini. *Follow me cloud: interworking federated clouds and distributed mobile networks*. IEEE Network, vol. 27, no. 5, pages 12–19, Sep. 2013.

[Tanaka 2018] Hiroyuki Tanaka, Masahiro Yoshida, Koya Mori and Noriyuki Takahashi. *Multi-access Edge Computing: A Survey*. Journal of Information Processing, vol. 26, pages 87–97, 01 2018.

[Tanenbaum 2015] A.S. Tanenbaum and H.J. Bos. Modern operating systems, 4th edition. Pearson Higher Education, 2015.

[Urgaonkar 2015] Rahul Urgaonkar, Shiqiang Wang, Ting He, Murtaza Zafer, Kevin Chan and Kin K. Leung. *Dynamic Service Migration and Workload Scheduling in Edge-clouds*. Perform. Eval., vol. 91, no. C, pages 205–228, September 2015.

[Varga 2008] A. Varga and R. Hornig. *An overview of the OMNeT++ simulation environment*. In Simutools'08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems, March 2008.

[Vazirani 2001] Vijay V. Vazirani. Approximation algorithms. Springer-Verlag, Berlin, Heidelberg, 2001.

[Veh] *Cellular and Connected Car Vehicles*. https://www.gsma.com/iot/news/connected-vehicles-are-safest-when-they-are-cellular/.

[VFC] *Vehicular Fog Computing*. https://medium.com/analytics-vidhya/vehicular-fog-computing-communication-without-lag-3c2452f4885b.

[Wang 2015] S. Wang, X. Zhang, J. Zhang, J. Feng, W. Wang and K. Xin. *An Approach for Spatial-Temporal Traffic Modeling in Mobile Cellular Networks*. In 2015 27th International Teletraffic Congress, pages 203–209, Sept 2015.

[Wang 2018a] L. Wang and E. Gelenbe. *Adaptive Dispatching of Tasks in the Cloud*. IEEE Transactions on Cloud Computing, vol. 6, no. 1, pages 33–45, 2018.

[Wang 2018b] S. Wang, J. Xu, N. Zhang and Y. Liu. *A Survey on Service Migration in Mobile Edge Computing*. IEEE Access, vol. 6, pages 23511–23528, 2018.

[was] *Web Assembly*. https://webassembly.org/.

[Willinger 1998] W. Willinger and V. Paxson. *Where Mathematics Meets the Internet*, 1998.

[Wolsey 1998] L.A. Wolsey. Integer programming. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.

[Xiao 2017a] Y. Xiao and Chao Zhu. *Vehicular fog computing: Vision and challenges*. In 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pages 6–9, 2017.

[Xiao 2017b] Yu Xiao, Marius Noreikis and Antti Ylä-Jääski. *QoS-oriented capacity planning for edge computing*. In IEEE International Conference on Communications, ICC 2017, Paris, France, May 21-25, 2017, pages 1–6, 2017.

[Xu 2016] Zichuan Xu, Weifa Liang, Wenzheng Xu, Mike Jia and Song Guo. *Efficient Algorithms for Capacitated Cloudlet Placements*. IEEE Transactions on Parallel and Distributed Systems, vol. 27, pages 2866–2880, 2016.

[Xu 2017] Zhigang Xu, Xiaochi Li, Xiangmo Zhao, Michael Zhang and Zhongren Wang. *DSRC versus 4G-LTE for Connected Vehicle Applications: A Study on Field Experiments of Vehicular Communication Performance*. Journal of advanced transportation, vol. 435, 08 2017.

[Yao 2015] Hong Yao, Changmin Bai, Deze Zeng, Qingzhong Liang and Yuanyuan Fan. *Migrate or Not? Exploring Virtual Machine Migration in Roadside Cloudlet-based Vehicular Cloud*. Concurr. Comput. : Pract. Exper., vol. 27, no. 18, pages 5780–5792, December 2015.

[Yousefpour 2018] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong and Jason P. Jue. *All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey*, 2018.

[Zhang 2018a] Fei Zhang, Guangming Liu, Xiaoming Fu and Ramin Yahyapour. *A Survey on Virtual Machine Migration: Challenges, Techniques, and Open Issues*. IEEE Communications Surveys and Tutorials, vol. 20, pages 1206–1243, 2018.

[Zhang 2018b] Qi Zhang, Ling Liu, Calton Pu, Qiwei Dou, Liren Wu and Wei Zhou. *A Comparative Study of Containers and Virtual Machines in Big Data Environment*. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pages 178–185, 2018.