Operation eco-energetique : le jeu du placement des microservices

Imane Taleb, Jean-Loup Guillaume, Benjamin Duthil

Operation eco-energetique : le jeu du placement des microservices

Imane Taleb[1], Jean-Loup Guillaume[1] et Benjamin Duthil[2]

[1]*L3i, La Rochelle Université, France*
[2]*EIGSI, La Rochelle, France*

Le développement des infrastructures Cloud-Fog-Edge conçu en réponse à l'avancée de l'IoT nécessite le positionnement des applications au plus près des utilisateurs. Les microservices permettent cela, mais exigent un placement adapté. Nous proposons ici une heuristique pour le placement de microservices dans le continuum Cloud-Fog-Edge, basée sur la détection de communautés pour optimiser la consommation d'énergie tout en prenant en compte les ressources des nœuds et en garantissant un temps de réponse acceptable. Les résultats, comparés à d'autres scénarios, montrent que notre approche peut réduire de manière significative la consommation d'énergie.

Mots-clefs : Placement de microservices, Cloud-Fog-Edge Continuum, efficacité énergétique, partitionnement de graphe.

1 Introduction

L'avancée des technologies intelligentes et de l'IoT a engendré de nouvelles applications, services et infrastructures réseau. Pour avoir des temps de réponse faibles il est nécessaire de déployer ces applications au plus près des utilisateurs et c'est ce qu'offre le continuum Cloud-Fog-Edge. Cependant, l'hétérogénéité de cette infrastructure pose des défis de mobilité des nœuds, de défaillances, de ressources limitées... Ces contraintes ont mené à l'adoption d'applications basées sur les microservices (MS), qui sont des modules légers et flexibles mais qui doivent communiquer entre eux. Un positionnement inadéquat peut entraîner une latence élevée ainsi qu'une augmentation des coûts et de la consommation d'énergie.

S'il y a de nombreux travaux sur le placement d'applications dans les environnements Fog pour optimiser la latence ou la qualité de service [PKB19], le placement des MS reste sous-exploré, en particulier en ce qui concerne la consommation d'énergie. Nous proposons ici une heuristique de placement basée sur la détection de communautés visant à optimiser la consommation d'énergie en minimisant les distances de communication entre les MS, tout en tenant compte des contraintes de ressources.

Après avoir passé en revue les recherches récentes dans la section 2, nous décrivons le modèle utilisé, notre heuristique et les résultats obtenus dans les sections 3 et 4, avant de conclure dans la section 5.

2 État de l'art synthétique sur le placement de microservices

Les méthodes de placement de MS utilisent majoritairement des méta-heuristiques, en particulier l'optimisation par essaims particulaires (PSO), ou ses versions distribuée ou discrète. C'est notamment le cas pour créer les différentes instances de MS [PKB19, YYG$^+$19] auxquelles s'ajoutent des méthodes d'équilibrage de charge pour répartir les demandes. On trouve également des méthodes basées sur l'optimisation de baleine (WOA) ou des approches Cuckoo Search.

D'autres approches utilisent explicitement la structure du réseau et de l'application MS, par exemple pour identifier les MS les plus centraux d'une application qui seront regroupés dans des conteneurs à haut rendement énergétique [SMO$^+$22]. Plusieurs approches vont plus loin en utilisant le partitionnement de graphes. Par exemple, dans [SSP21], les auteurs utilisent un graphe avec plusieurs couches, chacune représentant la similarité du réseau des nœuds du Fog, du processeur, de la mémoire et du stockage. La détection de communautés permet ensuite d'identifier des nœuds similaires pour effectuer le placement. Enfin, des chercheurs ont proposé des modèles mathématiques, notamment des approches de résolution exactes basées sur la Programmation Linéaire en nombres entiers ou mixtes dans les fonctions de réseau virtuel [ACG20].

3 Modèle, formulation du problème et heuristique

La couche physique est modélisée comme un graphe avec t nœuds d'exécution et des liens réseaux entre ces nœuds. On distingue trois types de nœuds (Cloud, Fog et Edge) avec des ressources de plus en plus limitées. On note $N = \{n_1, n_2, ..., n_t\}$ l'ensemble des nœuds, et chaque nœud $n_j \in N$ a les caractéristiques suivantes : une vitesse de calcul cpu_j en MIPS, une mémoire de ram_j GB et une consommation électrique allant de p_j^{idle} si le nœud n'est pas utilisé à p_j^{max} s'il est utilisé à pleine capacité avec une croissance linéaire entre ces deux valeurs. Chaque lien $l = \{n_i, n_j\}$ est caractérisé par une bande passante bw_l et une latence lc_l. On peut également considérer des liens logiques l' qui représentent les plus courts chemins en termes de latence, ainsi : $bw_{l'} = min_{l \in L}(bw_l)$ et $lc_{l'} = \sum_{l \in L}(lc_l)$.

Les applications que nous étudions sont composées d'un ensemble de MS, chacun ayant des besoins en ressources spécifiques : l'unité centrale, la bande passante et la mémoire vive. Ils communiquent via des appels de fonction (API). Une application peut ainsi être modélisée comme un graphe dirigé acyclique (DAG) dans lequel les nœuds représentent les MS $S = \{s_1, s_2, ..., s_t\}$. Chaque MS $s_i \in S$ consomme des ressources pour son exécution : du temps CPU cpu_i en million d'instructions et un espace mémoire ram_i en GB. Chaque arc $m = (s_i, s_j) \in M$ représente un appel de fonction entre les MS correspondant et est caractérisé par un transfert de données de taille $data_{i,j}$.

Nous définissons également le concept de chemin fonctionnel de MS, ou MFP (Microservice Function Path), qui désigne une séquence de dépendances nécessaires pour exécuter une tâche particulière. Un MFP contient un MS source et toutes ses dépendances directes et indirectes comme illustré sur la figure 1. Dans une architecture microservices, les nœuds sources servent de points d'entrée principaux pour les demandes des utilisateurs et tous les descendants de ces nœuds servent potentiellement à répondre la requête.

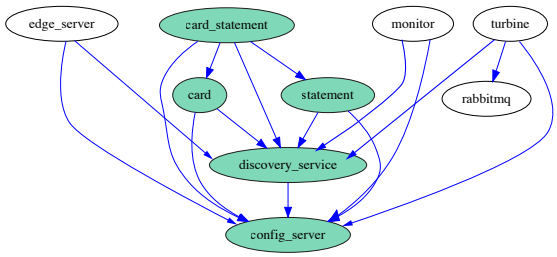


FIGURE 1 – Graphe de dépendances entre MS de l'application de relevé de carte d'un blog informatique, extrait de [RT19]. Un (des 4) MFP est mis en évidence sur ce graphe.

3.1 Formulation du problème

L'objectif est de placer les MS sur les nœuds du graphe de sorte à minimiser la consommation énergétique. Comme cela est fait dans la littérature [DSMP19] on considère une corrélation linéaire entre l'utilisation de ressources et la consommation d'énergie. La consommation énergétique d'un nœud j utilisé est p_j^{idle} auquel on ajoute $\left(p_j^{max} - p_j^{idle}\right) \frac{cpu_i}{cpu_j}$ pour chaque MS i placé dessus. La consommation d'une communication entre les MS i et j respectivement placés sur les nœuds k et l est $\left(\frac{data_{(i,j)}}{bw_{(k,l)}} + lc_{(k,l)}\right)\left[p_k^{max} + p_l^{max} - p_k^{idle} - p_l^{idle}\right]$.

3.2 Heuristique de placement de microservices

Dans un graphe, les communautés sont des groupes de nœuds densément connectés. Cela signifie qu'il y a beaucoup d'arêtes à l'intérieur d'une communauté et peu entre les communautés. Dans notre contexte, nous faisons l'hypothèse que la consommation d'énergie sera d'autant plus élevée que les MS sont sur des nœuds dans des communautés différentes. L'objectif est alors de placer des MFP complets au sein d'une même communauté dès lors qu'on ne peut pas les placer sur un seul nœud. L'heuristique commence donc par identifier des communautés [BGLL]. Chaque communauté obtient un score en fonction de ses

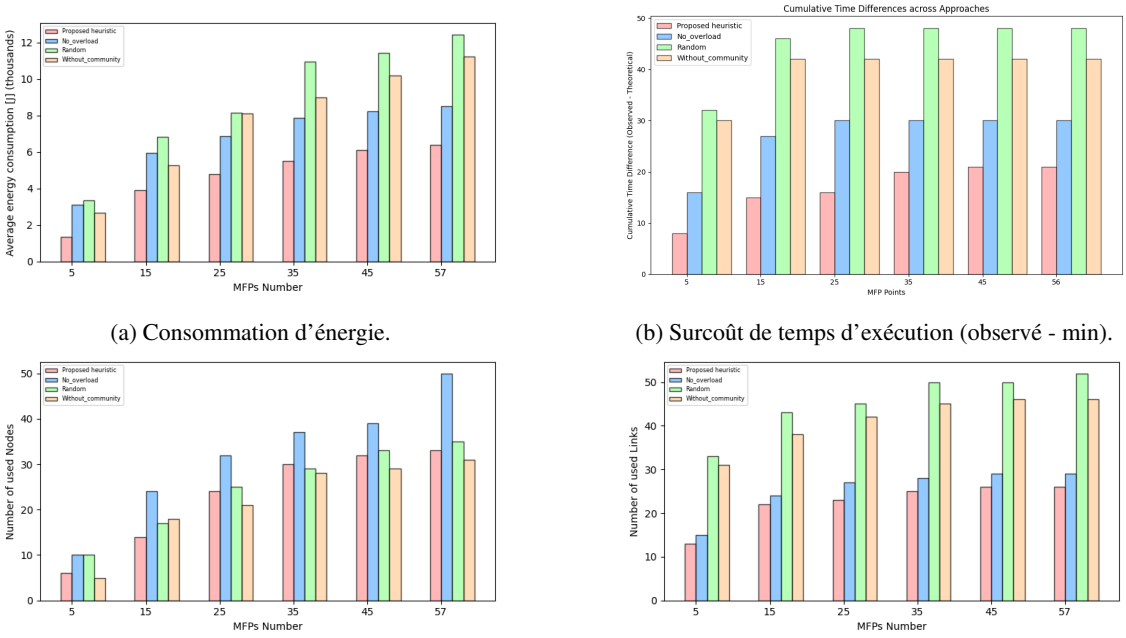
ressources et de sa taille et, de même, les MFP ont un score lié à leurs besoins. L'algorithme place ensuite le plus gros MFP dans la plus petite communauté, met à jour les ressources disponibles et itère.

Une fois les MFP associés aux communautés, nous utilisons l'algorithme de Kernighan-Lin [HL95] (KL) pour identifier différents groupes de MS. KL permet de bipartitionner un graphe en minimisant les nombres de liens entre les parties. On répète ce découpage pour construire un dendrogramme (binaire) complet. On sélectionne ensuite le nœud avec le plus de capacité et de plus fort degré et on place le plus de MS sur ce nœud en respectant le dendrogramme (c'est-à-dire en y plaçant des MS dans l'ordre d'un parcours préfixe). Une fois le nœud plein on place les autres MS sur le nœud suivant et ainsi de suite.

4 Évaluation et résultats

Nous nous assurons que tous les MS sont déployés en respectant les contraintes de ressources. Nous étudions ensuite la consommation d'énergie et le temps d'exécution de chaque MFP. Nous nous comparons à trois scénarios : **Pas de surcharge** qui utilise la même heuristique mais limite la charge sur chaque nœud à 70 % ; **Sans communautés** où on place les MFP comme dans l'heuristique mais sans considérer les communautés ; **Aléatoire** où l'on place itérativement les MFP sur des nœuds choisis au hasard.

Nos résultats utilisent la topologie Oteglobe de la base Internet ZOO topology [KNF+11] dont la composante géante contient 81 nœuds. Le nœud le plus central fait office de Cloud, puis 50% des autres nœuds les plus centraux sont considérés Fog et les autres sont Edge. Les caractéristiques des nœuds sont choisies avec une distribution uniforme aléatoire comme dans [SSP21]. Pour les MS nous avons utilisé les graphes de dépendances de [RT19] qui contient 20 applications avec un nombre de MS allant de 5 à 25. Nous avons également utilisé une distribution uniforme pour les besoins des MS et les tailles des messages.



(a) Consommation d'énergie.

(b) Surcoût de temps d'exécution (observé - min).

(c) Nombre de nœuds utilisés.

(d) Nombre de liens utilisés.

FIGURE 2 – Comparaison des différents scénarios de placement des microservices.

4.1 Résultats

Pour analyser et évaluer la performance de notre heuristique et la comparer aux autres scénarios, nous étudions plusieurs mesures : (1) la consommation d'énergie du placement en mesurant l'énergie dans les nœuds et celle des communications ; (2) le surcoût temporel engendré par le placement par rapport au temps

minimum théorique ; (3) le nombre de nœuds utilisés pour le déploiement de tous les MFP, en considérant qu'un nœud est utilisé si au moins un MS y est placé et (4) le nombre de liens utilisés.

Notre heuristique présente un net avantage en termes de consommation d'énergie (voir figure 2a). Notre approche réduit naturellement la consommation d'énergie par rapport à la stratégie qui évite la surcharge mais oblige à placer des MS dans un plus grand nombre de nœuds. L'écart est encore plus élevé comparativement à la version sans communautés qui ne place pas les nœuds dans des zones avec beaucoup de possibilités de connexions. Naturellement, la stratégie aléatoire est beaucoup moins efficace. Ces éléments sont confirmés par les figures 2c et 2d qui suggèrent une meilleure optimisation de l'allocation des ressources et des communications. La stratégie "Pas de surcharge" utilise plus de nœuds mais reste efficace en termes de liens. Les deux autres stratégies, "Sans communauté" et "Aléatoire", utilisent efficacement les nœuds du réseau mais induisent trop de communications. Enfin, cette efficacité énergétique et meilleure utilisation des ressources induit un meilleur respect des temps d'exécution comme illustré par la figure 2b.

5 Conclusion

Nous avons développé une heuristique pour le placement des MS qui identifie les nœuds du réseau proches et fortement connectés afin d'y placer les MFP. Nous avons ensuite placé les plus grands MFP dans les plus petites communautés pouvant les accueillir puis les MS des MFP sur les nœuds du réseau. Notre heuristique donne de meilleurs résultats en termes d'utilisation des nœuds et des liens, ainsi qu'une consommation d'énergie inférieure à celle de trois autres stratégies. Dans nos travaux futurs, nous souhaitons prendre en compte des instances multiples de MS et mettre en œuvre des stratégies de placement dynamique qui répondent aux besoins des utilisateurs en temps réel.

Références

[ACG20] B. Addis, G. Carello, and M. Gao. On a virtual network functions placement and routing problem : Some properties and a comparison of two formulations. *Networks*, 75(2) :158–182, 2020.

[BGLL] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics : theory and experiment*, 2008(10) :P10008.

[DSMP19] T. Djemai, P. Stolf, T. Monteil, and J.-M. Pierson. A discrete particle swarm optimization approach for energy-efficient iot services placement over fog infrastructures. In *18th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 32–40. IEEE, 2019.

[HL95] B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. *SC*, 95(28) :1–14, 1995.

[KNF+11] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9) :1765–1775, 2011.

[PKB19] S. Pallewatta, V. Kostakos, and R. Buyya. Microservices-based iot application placement within heterogeneous and resource constrained fog computing environments. In *12th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 71–81, 2019.

[RT19] MI. Rahman and D. Taibi. A curated dataset of microservices-based systems. In *Summer School on Software Maintenance and Evolution*. CEUR-WS, September 2019.

[SMO+22] A. Saboor, A.K. Mahmood, A.H. Omar, M.F. Hassan, S.N.M. Shah, and A. Ahmadian. Enabling rank-based distribution of microservices among containers for green cloud computing environment. *Peer-to-Peer Networking and Applications*, 15(1) :77–91, 2022.

[SSP21] Z.N. Samani, N. Saurabh, and R. Prodan. Multilayer resource-aware partitioning for fog application placement. In *5th IEEE International Conference on Fog and Edge Computing (ICFEC)*, pages 9–18, 2021.

[YYG+19] Y. Yu, J. Yang, C. Guo, H. Zheng, and J. He. Joint optimization of service request routing and instance placement in the microservice system. *Journal of Network and Computer Applications*, 147 :102441, 2019.