Optimisation multi-objectif

Gestion des ressources cloud : optimisation de l'allocation entre serveurs MEC et cloud

ING3 IAC Semaine A, CY-Tech
10 janvier 2026

Auteurs : Andre Victor
Bogaer Youenn
Dano Ewen

Résumé

La croissance massive de l'Internet des objets (IoT) a entraîné une explosion des flux de données, nécessitant un traitement distribué entre serveurs MEC et cloud. Réduire la latence tout en équilibrant la charge et en maîtrisant la consommation énergétique constitue un enjeu majeur. Ce rapport présente une modélisation mathématique permettant l'optimisation conjointe du temps d'exécution, de la répartition de charge et du coût énergétique.

Problématique : Comment allouer les ressources de calcul entre les serveurs MEC et le cloud de manière à minimiser simultanément le temps d'exécution, le déséquilibre de charge et la consommation énergétique ?

Notre travail se fonde sur l'article suivant : *Multi-objective resource allocation in mobile edge computing using PAES for Internet of Things* (Qi Liu, Ruichao Mao, Xiaolong Xu & Xu Ma).

1 Introduction

1.1 Contexte

Le Edge Computing consiste à allouer la puissance de calcul la plus proche de l'utilisateur, pour minimiser le temps de réponse et optimiser la bande passante.

L'énorme croissance du nombre d'objets connectés (IoT) et de leur puissance de calcul sature les systèmes réseaux de données toujours plus nombreuses et plus complexes. Quand le Cloud gère les demandes sur des serveurs distants, le Edge computing utilise les ressources de calculs les plus proches possibles physiquement. Tenter de traiter les demandes localement avant d'éventuellement les transférer vers le Cloud.

Plusieurs echelles d'Edge Computing :

— Network Edge : un datacenter à quelque kilomètres de l'utilisateur.
— On-Premise Edge : une box internet, un serveur LAN à quelque mettres.
— Device Edge : l'appareil qui capte la donnée.

Le Multi-access Edge Computing (MEC) est le paradigme moderne pour le déploiement du Edge Computing. Les serveurs MEC agissent à l'échelle du Network Edge en orchestrant les ressources du réseau local en un cloud distribué et permettant la communication fluide entre le réseau et les applications de l'IoT. Enfin le standard MEC assure la cohésion des différents équipements et des types de connexions (mobile 5G, Wifi...).

Cependant les capacités limitées des nœuds edge obligent parfois à déléguer une partie du traitement vers d'autres MEC ou vers le cloud. Les applications de l'IoT rencontrent des difficultés à trancher entre les différentes contraintes rencontrées.

1.2 Objectif

L'objectif est de déterminer une allocation optimale des ressources pour minimiser conjointement le temps d'exécution des tâches, le déséquilibre de charge et la consommation énergétique.

— Temps total : on souhaite minimiser le temps total d'exécution des tâches.
— Énergie consommée : on souhaite minimiser l'énergie consommée par l'opération.
— Équilibrage des charges : on souhaite équilibrer la sollicitation des serveurs MEC. La consommation énergétique et la température des serveurs augmentent avec l'utilisation.

2 Notations et hypothèses

— app_n : tâche à exécuter, $n \in [\![1, N]\!]$ avec $N \in \mathbb{N}^*$.
— MEC_m : serveur MEC, $m \in [\![1, M]\!]$ avec $M \in \mathbb{N}^*$.
— $MEC(n)$: le serveur MEC "parent" de app_n (le plus proche).
— $x_{n,m} \in \{0, 1\}$: 1 si la tâche n est exécutée sur le serveur m.

Nous désignons la puissance de chaque serveur MEC en nombre de CPU.

— $V_m \in \mathbb{N}^*$: capacité maximale du serveur m (en unités CPU).
— $U_n \in \mathbb{N}^*$: nombre CPU allouées à la tâche n.

Nous supposons qu'une tâche est exécutée entièrement sur un seul et même serveur.

Soit $n \in [\![1, N]\!]$,

— On définit $MEC_{exe}(n) \in [\![0, M]\!]$ le serveur MEC associé à la tâche n.
— Note : $MEC_{exe}(n) = 0$ si la tâche n n'est affectée à aucun MEC mais au cloud. Alors on définit $U_{NC}(n) = U_n \cdot H(MEC_{exe}(n))$ qui désigne le nombre de CPU alloués au tâches effectuées sur un MEC (non cloud) avec H la fonction de Heaviside

3 Modélisation mathématique

3.1 Coût temporel

Pour chaque tâche app_n, le temps total est :

$$T_n = T_{1n} + T_{2n} + T_{3n}.$$

La tâche peut être exécutée à différents niveaux :

$$j \in \{0 : \text{local MEC},\ 1 : \text{inter-MEC},\ 2 : \text{cloud}\}.$$

Bandes passantes en $Mb \cdot s^{-1}$:

$$BP_p : \text{local}, \quad BP_q : \text{inter-MEC}, \quad BP_r : \text{MEC–cloud}.$$

Temps de transmission de la requête :

Avec G_n la taille de la tâche en $Mbits$ et RG_n la taille du résultat.

$$T_{1n} = \begin{cases} \frac{G_n}{BP_p} & j = 0 \\ \frac{G_n}{BP_p} + \frac{G_n}{BP_q} & j = 1 \\ \frac{G_n}{BP_p} + \frac{G_n}{BP_q} + \frac{G_n}{BP_r} & j = 2 \end{cases}$$

Temps de transmission du résultat :

$$T_{3n} = \begin{cases} \frac{RG_n}{BP_p} & j = 0 \\ \frac{RG_n}{BP_p} + \frac{RG_n}{BP_q} & j = 1 \\ \frac{RG_n}{BP_p} + \frac{RG_n}{BP_q} + \frac{RG_n}{BP_r} & j = 2 \end{cases}$$

On remarque que T_{1n} et T_{3n} sont :

— minimal quand j = 0, i.e. $MEC_{exe}(n) = MEC(n)$.
— maximal quand j = 2, i.e. la tâche est exécutée sur le cloud.

Note : une tâche n est exécutée sur le cloud si $U_n \geq max_{m \in [\![1,M]\!]} V_m$

Temps de calcul du résultat :

Avec P_u la performance unitaire d'une VM en $Mb \cdot s^{-1}$:

$$\forall n \in [\![1, N]\!], \qquad T_{2n} = \frac{G_n}{U_n * P_u}..$$

Temps total global :

$$T_{\text{tot}} = \max_{n=1}^{N} T_n.$$

3.2 Consommation énergétique

Hypothèse : On suppose qu'on ne compte pas la consommation énergétique des tâches effectuées sur le cloud.

Coût énergétique des serveurs à vide en $kW \cdot h$ (on suppose les serveurs constamment allumés) :

$$EC_v = M \cdot r_v \cdot T_{tot},$$

où r_v désigne la puissance électrique d'un serveur à vide en W.

Coût énergétique lié d'un CPU alloué :

Sous l'hypothèse qu'une tâche ne s'exécute que sur un seul MEC ou sur le cloud,

$$EC_o = r_o \cdot \sum_{n=1}^{N} U_{NC}(n) \cdot T_{2n},$$

où r_o désigne la puissance électrique d'une CPU occupé.

Coût énergétique lié aux CPU non alloués :

$$EC_u = r_u \cdot [\sum_{m=1}^{M} V_m \cdot T_{tot} - \sum_{n=1}^{N} U_{NC}(n) \cdot T_{2n}], \tag{1}$$

où r_u désigne la puissance électrique d'une CPU inutilisé. Preuve fournie en annexe.

Consommation totale :

$$EC_{\text{tot}} = EC_v + EC_o + EC_u.$$

3.3 Répartition des charges

Pour un équilibre optimal, on souhaite réduire l'hétérogénéité des utilisations de serveurs.

Sous l'hypothèse qu'une tâche est exécutée entièrement sur une seul MEC, on définit la quantité de CPU utilisé sur un serveur : $\nu_m = \sum_{n=1}^{N} U_n x_{n,m}, \quad \forall m \in [\![1, M]\!]$.

On utilise la variance :

$$MSU = \frac{1}{M} \sum_{m=1}^{M} (\nu_m - \bar{\nu})^2,$$

où $\bar{\nu}$ est l'utilisation moyenne :

$$\bar{\nu} = \frac{1}{M} \sum_{m=1}^{M} \nu_m.$$

4 Contraintes

Domaines des variables :

$$x_{n,m} \in \{0, 1\}, \quad U_n \in \mathbb{N}^*, \quad MEC_{exe}(n) \in [\![0, M]\!]$$

Chaque tâche doit être exécutée sur un unique serveur ou sur le cloud :

$$\sum_{m=1}^{M} x_{n,m} \leq 1, \quad \forall n.$$

Capacité des serveurs :

$$\nu_m \leq V_m, \quad \forall m.$$

Ressources minimales si la tâche est presente sur le serveur, ressources nulles si absente :

$$x_{n,m} \leq U_n x_{n,m} \leq V_m x_{n,m}, \quad \forall n, m$$

5 Fonctions objectifs

$$F_1 = T_{\text{tot}} \quad \text{(minimisation du temps total)}$$

$$F_2 = EC_{\text{tot}} \quad \text{(minimisation de l'énergie consommée)}$$

$$F_3 = MSU \quad \text{(équilibrage des charges)}$$

Problème global :

$$\min(F_1,\ F_2,\ F_3).$$

6 Modèle

6.1 Encodage : définition du chromosome

Les solutions que nous allons rechercher devront optimiser les trois fonctions objectives. Le choix que doit illustrer une solution est :

— Quelle répartition pour les tâches entre les serveurs et le cloud ?
— Combien de ressources allouer à chaque tâches ?

Définition du chromosome Soit $n \in [\![1, N]\!]$, on définit

$$c_n = \begin{bmatrix} MEC_{exe}(n) \\ U_n \end{bmatrix}$$

— $MEC_{exe}(n) \in [\![0, M]\!]$ le choix du serveur MEC pour la tâche n
— $U_n \in \mathbb{N}$ le nombre de VM allouées à la tâche n

Chromosome

$$C = \left[c_n\right]_{n \in [\![1,N]\!]}$$

Mutations

— Changer le serveur : $MEC_{exe}(n)$
— Augmenter ou diminuer les ressources : U_n

6.2 Solveur : PAES (Pareto Archived Evolution Strategy)

PAES est un algorithme d'optimisation à mémoire locale une mémoire externe (archive) introduit par J. Knowles et D. Corne dans l'article : *The Pareto Archived Evolution Strategy : A New Baseline Algorithm for Pareto Multiobjective Optimisation.*

L'algorithme se base sur de la recherche locale (1 + 1), en se concentrant sur une solution courante et une solution candidate (mutation de la solution courante). Contrairement à d'autres algorithmes qui travaillent sur des populations de solutions. Pour effectuer l'optimisation multi objectif, on entretient une archive des précédentes solutions non dominées.

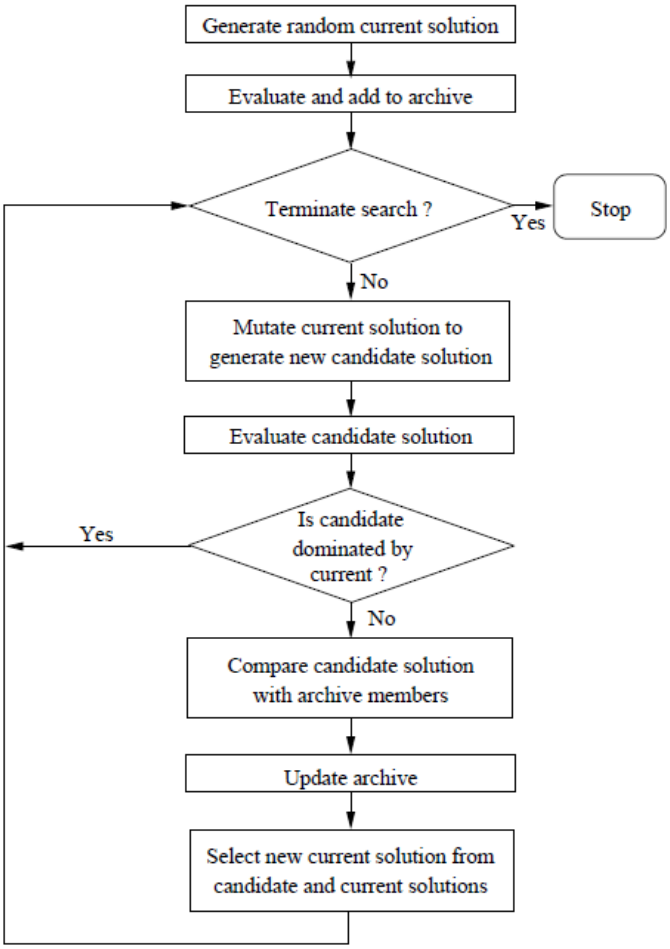

Figure 1: The Pareto Archived Evolution Strategy

L'algorithme se compose de trois éléments.

— Le générateur de solutions candidates
— La fonction d'acceptation des solutions candidates.
— L'archive des solutions non dominées (NDS : non dominated solutions).

6.2.1 Le générateur de solutions candidates

Créer un nouveau chromosome par mutation de la solution courante. Ce nouveau chromosome, si valide, devient la solution candidate.

6.2.2 La fonction d'acceptation des solutions candidates

L'acceptation est trivial si l'une des solutions domine l'autre ; la solution dominée est rejetée, l'autre est acceptée.

Si aucune des solutions ne domine, le candidat est comparé avec la population de référence contenue dans l'archive NDS. Si le mutant domine une ou plusieurs solutions de l'archive, il est accepté et ces solutions sont rejetées.

Si la comparaison avec la population échoue à trouver un dominant ou un dominé pour la solution candidate, et si l'archive est pleine alors on choisit la solution, parmi la courante et la candidate, qui se trouve dans la région la moins densément peuplée de l'espace. Si l'archive n'est pas pleine on ajoute simplement la solution candidate.

6.2.3 L'archive des solutions non dominées NDS

D'abord, l'archive NDS garde en mémoire l'ensemble des solutions non dominées générées. Ce qui servira pour retourner la réponse de l'algorithme

Ensuite elle sert à sélectionner les meilleures solutions parmi les mutants générés.

Enfin, l'archive assure une bonne répartition homogène, des solutions sur le front de Pareto. Pour se faire on divise l'espace en une grille 3D et compte le nombre de solutions pas bloc.

On définit un nombre maximum d'éléments dans l'archive, $refpop$, qui correspond au nombre de solutions finales désiré.