Placement et migration de conteneurs dans le cloud : vers un modèle MILP modulaire et ses briques d'optimisation

Timon Roxard, Etienne Leclercq, Jonathan Rivalan

Smile

{timon.roxard, etienne.leclercq, jonathan.rivalan}@smile.fr

Mots-clés : *Orchestration de conteneurs, optimisation combinatoire, MILP, cloud*

1 Introduction

Les plateformes d'orchestration de conteneurs, telles que Kubernetes, sont devenues un standard pour déployer des applications dans le cloud [1]. Elles décident en continu du *placement* de conteneurs sur des nœuds hétérogènes et de leur éventuelles *migrations* en fonction de la charge, des pannes, des coûts ou de la réglementation. Selon le contexte (cloud privé / public / hybride, edge, fog), les opérateurs doivent concilier disponibilité, consommation énergétique, coûts, latence et conformité réglementaire, pour des échelles allant de quelques conteneurs à des centaines de milliers. De nombreuses stratégies de placement et de migration ont été proposées, en particulier sous forme d'heuristiques, et plusieurs revues récentes en dressent une synthèse détaillée [2]. Cependant, aucun de ces travaux ne propose un cadre de modélisation unifié pour le placement et la migration de conteneurs, permettant de combiner plusieurs exigences et de raisonner sur la complexité. Nous présentons ici un **modèle générique de type MILP** pour le placement et la migration qui puisse servir de socle à de nombreuses problématiques d'orchestration. Ce modèle prolonge les formulations MILP utilisées pour l'allocation de ressources cloud [3] et les approches de consolidation énergétique par migration [4], tout en les adaptant à la variété des contextes conteneurs grâce à une structure modulaire, ainsi qu'en introduisant une dimension temporelle.

2 Modèle générique de placement et migration

On considère un ensemble de nœuds N, un ensemble de conteneurs C et un horizon discret T. À chaque date, chaque conteneur est hébergé sur un nœud. Le problème consiste à décider, sur l'horizon T, du placement des conteneurs et des migrations. Le modèle ci-dessous fournit un socle minimal pour la structure temporelle et les contraintes de capacité, que l'on enrichira ensuite par briques d'orchestration. Il s'agit d'une variante de *bin packing temporel*, où la taille des objets (conteneurs) évolue dans le temps et où les déplacements (migrations) sont autorisés. L'équation 1 s'assure qu'un conteneur est sur un (et un seul) nœud, l'équation 2 s'assure que la capacité des nœuds permet de contenir les conteneurs, et l'équation 3 assure qu'un conteneur ne peut changer de nœud que lorsqu'une migration est autorisée.

Variables.

$x_{n,c,t} \in \{0,1\}$: conteneur c sur nœud n à t,
$m_{c,t} \in \{0,1\}$: migration de c entre t et $t+1$.

Paramètres.

$r_{c,t} \in \mathbb{R}^+$: taille du conteneur c à la date t,
$R_n \in \mathbb{R}^+$: capacité totale du nœud n.

Contraintes.

$$\sum_{n \in N} x_{n,c,t} = 1 \qquad \forall c \in C,\ \forall t \in T, \tag{1}$$

$$\sum_{c \in C} r_{c,t}\, x_{n,c,t} \leq R_n \qquad \forall n \in N,\ \forall t \in T. \tag{2}$$

$$x_{n,c,(t+1)} \geq x_{n,c,t} - m_{c,t} \qquad \forall n \in N,\ \forall c \in C,\ \forall t \in T. \tag{3}$$

Objectif. (exemple)

$$\min \sum_{c,t} \alpha_c m_{c,t}, \tag{4}$$

où α_c est le coût de migration pour le conteneur c. Cet objectif minimise le coût total des migrations, mais d'autres termes peuvent être ajoutés selon le contexte, comme détaillé dans la section suivante.

3 Briques d'orchestration et impact sur la complexité

Nous appelons *brique d'orchestration* un ensemble cohérent de contraintes et de variables ajouté au modèle de base pour capturer une problématique opérationnelle : affinité, énergie, latence, etc. La Table 1 illustre les briques d'orchestration communes et leur impact qualitatif sur la taille du modèle.

Certaines briques, comme les affinités ou la compatibilité, ajoutent surtout des contraintes. D'autres, comme la migration ou le brownout, élargissent l'espace des placements réalisables. Toutes les briques d'orchestration envisagées peuvent être exprimées sous forme linéaire (LP/MILP). Elles restent donc compatibles avec le modèle générique et les solveurs standards, même si leur combinaison peut faire croître rapidement la complexité, rendant la résolution exacte difficile à grande échelle.

Brique	**Intention métier**	**Impact qualitatif**
Disponibilité	Tolérance aux pannes et maintenances.	Contraintes de vidage forcé de nœuds et de robustesse aux pannes, en $O(\|C\|\|T\|)$, sans nouvelles variables.
Affinités / anti-affinités	Regrouper ou séparer des conteneurs.	Contraintes de colocalisation (ou exclusion) entre couples de conteneurs, en $O(\|C\|^2\|T\|)$, sans nouvelles variables.
Scalabilité	Adapter le nombre de nœuds.	Variables binaires pour l'activation/désactivation des nœuds, et contraintes associées, en $O(\|N\|\|T\|)$.
Migration contrôlée	Limiter le nombre de migrations.	Contraintes limitant les migrations simultanées de conteneurs d'un même service, sans nouvelles variables.
Latence réseau	Respecter les budgets de latence.	Contraintes et variables binaires dépendantes de la topologie représentée, en $O(\|N\|^2\|C\|^2\|T\|)$.
Compatibilité	Restrictions de certains placements.	Contraintes d'incompatibilité nœud-conteneur, en $O(\|C\|\|T\|)$, sans nouvelles variables.
Brownout	Désactiver des services en cas de surcharge.	Relaxation de la contrainte de placement 1 (pénalités dans l'objectif), sans nouvelles variables ni contraintes.

TAB. 1 – Exemples de briques d'orchestration et impact qualitatif.

Lorsque seules certaines briques sont activées (par exemple capacité, scalabilité et affinités), le problème se rapproche de formulations exactes de scheduling de clusters de conteneurs déjà étudiées dans la littérature [5]. À l'inverse, l'activation simultanée, sur un cluster de taille importante, de contraintes de latence fine et d'anti-affinités, justifie le recours à des heuristiques ou à des approches de décomposition, comme c'est souvent le cas dans la pratique [1, 2].

Notre modèle générique fournit un *cadre commun* pour analyser les problématiques d'orchestration de conteneurs. Il permet d'identifier des **sous-problèmes tractables** (en activant un sous-ensemble de briques), d'anticiper les situations où des **heuristiques** ou des méthodes de **décomposition** sont nécessaires, et de rapprocher les politiques implémentées dans des orchestrateurs comme Kubernetes [1], des formulations exactes de la littérature [2, 3, 5]. Ces briques, leurs interactions et leur résolution ouvrent des perspectives, notamment sur la complexité induite par leurs combinaisons, l'utilisation de techniques du domaine de la commande optimale, ou l'intégration de critères QoS/QoE dans les objectifs.

Références

[1] K. Senjab, S. Abbas, N. Ahmed, A. ur R. Khan. A survey of Kubernetes scheduling algorithms. *J. Cloud Computing*, 12(1) :87, 2023.

[2] K. Kaur, F. Guillemin, F. Sailhan. Container placement and migration strategies for Cloud, Fog and Edge data centers : A survey. *Int. J. Network Management*, 32(6) :e2212, 2022.

[3] T. Guérout, Y. Gaoua, C. Artigues, G. Da Costa, P. Lopez, T. Monteil. Mixed integer linear programming for quality of service optimization in Clouds. *Future Gener. Comput. Syst.*, 71 :1–17, 2017.

[4] A. Beloglazov, R. Buyya. Energy Efficient Resource Management in Virtualized Cloud Data Centers. In : *Proc. IEEE/ACM CCGRID*, pp. 826–831, 2010.

[5] A. Asensio, X. Masip-Bruin, J. Garcia, S. Sánchez. On the optimality of Concurrent Container Clusters Scheduling over heterogeneous smart environments. *Future Gener. Comput. Syst.*, 118 :157–169, 2021.