Ordonnancement de conteneurs dans Kubernetes

Tarek Menouer

Umanis Research & Innovation
7 Rue Paul Vaillant Couturier,
Levallois-Perret, France,
tmenouer@umanis.com

Mots-clés : *Conteneurs, Ordonnancement multicritère, Optimisation de performance.*

1 Résumé

Ces dernières années, diverses technologies de virtualisation ont vu le jour dans le cloud computing, telles que les machines virtuelles (VM) et les conteneurs ayant chacun leurs propres caractéristiques. Un conteneur est une technique de virtualisation légère au niveau du système d'exploitation permettant d'exécuter une application et ses dépendances dans un processus isolé. Actuellement la conteneurisation est devenue très répandue, en raison de différents avantages tels que la facilité et la rapidité du déploiement.

Dans la litterature, plusieurs frameworks d'orchestration et d'ordonnancement de conteneurs sont proposés comme Docker SwarmKit [7], Apache Mesos [6] et Google Kubernetes [8].

Kubernetes est un framework open source trés utilisé dans le domaine industriel et académique. Il permet d'automatiser l'affectation des conteneurs aux nœuds de l'infrastructure [5, 2]. Kubernetes est basé sur une architecture maître-esclave où la communication entre le maître et les esclaves se fait à l'aide d'un mécanisme qu'il s'appelle kubelet [1].

Au niveau d'ordonnancement de conteneurs, plusieurs stratégies sont proposées dans la littérature. Le principe général consiste à sélectionner, pour chaque nouveau conteneur soumis par un utilisateur, un nœud parmi l'ensemble des nœuds qui constituent l'infrastructure Cloud. Cependant, la majorité des stratégies d'ordonnancement de conteneurs sont basées uniquement sur un seul critère afin de sélectionner un nœud comme : (i) le nombre de CPUs disponibles, (ii) la taille de la mémoire disponible, ou (iii) le nombre de conteneurs en cours d'exécution dans chaque nœud. La sélection mono-critère d'un nœud limite la performance car l'ordonnanceur a une vision limitée de l'état de l'infrastructure Cloud et les besoins des utilisateurs. Pour répondre à cette problématique, nous proposons une nouvelle stratégie d'ordonnancement de conteneurs dans le framework Kubernetes appelée KCSS (Kubernetes Container Scheduling Strategy) [4].

La nouveauté de KCSS est qu'elle considère plusieurs critères afin d'optimiser l'ordonnancement d'un ensemble de conteneurs soumis en ligne par des utilisateurs. L'objectif est de réduire le makespan (temps nécessaire pour ordonnancer et exécuter les conteneurs) et la consommation énergétique globale de l'infrastructure Cloud.

Afin de sélectionner, pour chaque conteneur, le meilleur nœud qui a un bon compromis entre plusieurs critères, nous proposons d'utiliser l'algorithme de décision multicritère TOPSIS (Technique for the Order of Prioritisation by Similarity to Ideal Solution) [3]. L'algorithme TOPSIS consiste à agréger tous les critères dans un seul rang. Ensuite, sélectionner le nœud qui a le rang le plus élevé.

Dans notre approche, nous considérons des critères hybrides liés à l'état de l'infrastructure Cloud et aux besoins des clients. Les critères considérés sont :

— Le taux d'utilisation de CPUs dans chaque nœud ;
— Le taux d'utilisation de la mémoire dans chaque nœud ;
— Le taux d'utilisation de l'espace de stockage dans chaque nœud ;

— La consommation énergétique de chaque nœud ;
— Le nombre de conteneurs en cours d'exécution dans chaque nœud ;
— Le temps de transmission de l'image sélectionnée par l'utilisateur pour son conteneur.

Pour une raison de simplicité, nous considérons dans notre étude uniquement six critères. Cependant, il est possible de généraliser le nombre de critères en considérant, par exemple, des critères liés au réseau comme la bande passante.

Les principales contributions de notre étude sont résumées comme suit :

1. Proposer une nouvelle stratégie d'ordonnancement de conteneurs dans Kubernetes appelée KCSS ;
2. Considérer dans KCSS des critères hybrides liés à l'état de l'infrastructure Cloud et aux besoins des utilisateurs ;
3. Appliquer un algorithme d'analyse de décision multicritères dans KCSS afin de sélectionner pour chaque conteneur le nœud qui a un bon compromis entre plusieurs critères ;
4. Implémenter KCSS en langage Go dans Kubernetes [8] avec un minimum de changement pour être utilisé facilement avec les prochaines versions de Kubernetes ;
5. Évaluer les performances obtenues avec KCSS et les comparer aux performances obtenues avec d'autres stratégies d'ordonnancement de conteneurs sous différents scénarios en termes de makespan et de la consommation énergétique.

Références

[1] Y. Biran, S. Pasricha, G. Collins, and J. Dubow. Enabling green content distribution network by cloud orchestration. In *2016 3rd Smart Cloud Networks Systems (SCNS)*, pages 1–8, 2016.

[2] S. Hoque, M. S. d. Brito, A. Willner, O. Keil, and T. Magedanz. Towards container orchestration in fog computing infrastructures. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 294–299, July 2017.

[3] Young-Jou Lai, Ting-Yun Liu, and Ching-Lai Hwang. Topsis for modm. *European Journal of Operational Research*, 76(3) :486 – 500, 1994. Facility Location Models for Distribution Planning.

[4] Tarek MENOUER. Kcss : Kubernetes container scheduling strategy. In *The Journal of Supercomputing*, 2020.

[5] René Peinl, Florian Holzschuher, and Florian Pfitzer. Docker cluster management for the cloud - survey results and own solution. *Journal of Grid Computing*, 14(2) :265–282, Jun 2016.

[6] The apache software foundation. mesos, apache : `http://mesos.apache.org/`, visited 16-03-2021.

[7] Docker swarm kit :
`https://github.com/docker/swarmkit/`, visited 16-03-2021.

[8] Kubernetes framework :
`https://kubernetes.io/`, visited 16-03-2021.