État de l'art des unikernels

Ludovic Muller

Mercredi 13 février 2019

1 Introduction

Aujourd'hui de plus en plus de services sont proposés aux utilisateurs. Le nombre de ressources à traiter est également fortement croissant et c'est ainsi que de nouveaux centres de données sont construits au fur et à mesure partout dans le monde.

Au départ, une machine physique était attribuée à un service particulier. Lorsque la demande augmentait, on ajoutait simplement une nouvelle machine dédiée uniquement à ce service.

Cependant le nombre de services proposés ne cessant de croître, il n'est plus possible de se contenter de ne faire tourner qu'un seul service par machine, notamment pour des raisons budgétaires et pour un déploiement beaucoup plus rapide.

Les fournisseurs de services souhaitent gagner en qualité pour faire face à la concurrence. Cette qualité se traduit notamment par une démultiplication des données à travers le globe : cela rapproche le contenu des utilisateurs finaux, diminue les temps de latence, et permet d'être plus robuste en cas de panne. En effet il y aura toujours une machine présente pour prendre le relais. On gagne donc en souplesse en étant capable d'adapter l'offre et la demande en temps réel.

Pour ces fournisseurs, pouvoir faire tourner un grand nombre de services sur une même machine permettrait d'économiser le nombre de machines et de mieux les rentabiliser, du fait que l'on utilisera pleinement les ressources d'une machine au lieu de n'utiliser qu'un faible pourcentage sur plusieurs machines.

La virtualisation répond parfaitement à ces attentes. En effet, virtualiser plusieurs systèmes sur une seule machine permettrait d'exécuter indépendamment un grand nombre de services.

La virtualisation répond à un besoin d'isolation. En effet, si l'on souhaite lancer un grand nombre de services différents sur une même machine physique, un service ne doit pas compromettre un des autres services tournant sur la machine. Les systèmes d'exploitation traditionnels répondent déjà en partie à ce besoin

d'isolation, en virtualisant la mémoire et en isolant le matériel avec la séparation de l'espace utilisateur de l'espace système via les appels système. En virtualisation de système d'exploitation, le besoin d'isolation va encore plus loin puisqu'il faut isoler la mémoire de chacun des systèmes d'exploitation ainsi que les opérations de lecture et d'écriture. Virtualiser ainsi un système d'exploitation est un peu comme si l'on créait des machines virtuelles à l'intérieur d'une seule et même machine physique ; c'est la raison pour laquelle on parle justement de machines virtuelles, ou VM pour *virtual machines* en anglais.

Enfin, on peut dire que la virtualisation offre une certaine sécurité, du fait de cette isolation, et qu'il est plus facile de gérer une machine virtuelle compromise qu'une machine physique; on peut en recréer une plus rapidement en clonant une VM de base par exemple. Virtualiser offre également une certaine tolérance aux pannes, puisqu'il est possible de dupliquer ou migrer très simplement des machines virtuelles qui tourneraient sur un matériel défaillant par exemple. Enfin, certaines entreprises ont besoin de faire tourner certaines applications obsolètes (applications *legacy*) : utiliser la virtualisation permettrait à ces entreprises de continuer à utiliser ces applications sans sacrifier la sécurité de leur parc informatique. On constate donc que la virtualisation offre une réelle souplesse dans la gestion des services.

Aujourd'hui, d'autres solutions que la virtualisation de systèmes d'exploitation traditionnels ont émergé [1,2]. On utilise par exemple de plus en plus le mécanisme des conteneurs : au lieu de virtualiser complètement un système d'exploitation, on fait tourner directement l'application sur la machine hôte, en interceptant les appels systèmes, offrant ainsi une alternative beaucoup plus légère et rapide que la virtualisation classique.

2 Définition du problème

La virtualisation offre une réelle souplesse comme nous avons pu le voir. Cependant, virtualiser un système d'exploitation traditionnel peut s'avérer plutôt lourd. Pour éviter cette lourdeur, le mécanisme des conteneurs peut être particulièrement efficace. Puisqu'ils tournent directement sur la machine hôte, on économise toute la couche de la virtualisation d'un système complet. Les ressources nécessaires en matière de stockage, de mémoire et de calcul sont moindres par rapport à l'utilisation massive de machines virtuelles : on peut donc faire tourner un nombre nettement plus élevé de services.

Cependant l'utilisation des conteneurs peut s'avérer problématique en termes de sécurité, notamment du fait qu'ils tournent directement sur l'hôte [1] et que le nombre d'appels système ne cesse de croître au fil des années [3] : aujourd'hui un noyau Linux en compte environ 400. L'API des appels système permet aux conteneurs d'interagir avec le système d'exploitation hôte et offre une gestion des processus, des *threads*, de la mémoire, du réseau, du système de fichiers, de

la communication IPC, etc. Le fait que les conteneurs tournent directement sur l'hôte nécessite également une homogénéité entre le système d'exploitation hôte et invité, ce qui peut être limitant.

Ce que l'on souhaiterait, c'est d'une part avoir la possibilité d'une isolation forte qui garantit une certaine sécurité, et d'autre part, d'être léger. Ceci permet un déploiement en masse de manière adaptative, une rapidité pour servir et s'adapter continuellement à la demande et ainsi faire face à la concurrence, tout en offrant une qualité de service sûre.

Cela fait quelques années que des équipes de chercheurs se posent la question et une solution semble émerger : les unikernels.

Que sont les unikernels ? En quoi diffèrent-ils des solutions actuellement utilisées ? Comment arrivent-ils à offrir des gains réels en performances sans sacrifier la sécurité, chose que l'on n'arrivait pas à satisfaire simultanément jusqu'à présent avec des conteneurs et des VM traditionnelles ?

3 Unikernels

Les unikernels pourraient être décrits simplement comme étant un système d'exploitation créé à partir d'un assemblage de briques de LEGO, chacune de ces briques étant une bibliothèque élémentaire permettant une tâche de base, comme par exemple la partie réseau, ou bien la communication IPC, etc. Un système construit de cette manière à partir de bibliothèques de base s'appelle une *library OS* ou *libOS*.

Le principe des unikernels est qu'au lieu de lancer un système d'exploitation classique composé d'un grand nombre d'applications, il va se charger de ne faire tourner que le binaire d'une application, et donc de ne faire tourner qu'un seul processus [2]. On va ainsi uniquement utiliser les briques dont l'application a réellement besoin pour fonctionner.

Pour éviter d'avoir à supporter l'ensemble des périphériques possibles, dans les cas des unikernels on va partir du principe qu'il tournera dans une machine virtuelle, et que ce sera à l'hyperviseur de s'occuper du matériel et d'en faire l'abstraction. Un hyperviseur est un programme qui permet la gestion (création, lancement, ...) des VM.

On se retrouve donc avec une image extrêmement légère, et comme il y a un lien de corrélation entre la taille des images et le temps de boot du fait du temps de chargement de l'ensemble en mémoire [3], ce qui fait que les unikernels peuvent démarrer beaucoup plus rapidement que les systèmes traditionnels. De plus, ils garantissent davantage de sécurité : il n'est même pas possible de se connecter sur la machine, puisqu'ils tournent directement au sein d'une machine virtuelle et n'incluent que le strict nécessaire pour faire tourner l'unique application. En outre, ils ne dépendent que d'un nombre très restreint de bibliothèques, limitant

le nombre de failles et bugs possibles, ce qui limite fortement la surface d'attaque [3].

Cependant, obtenir ces gains de performances tout en garantissant une certaine sécurité nécessite beaucoup de temps de configuration, étant donné qu'il faut spécialiser le plus possible le système pour l'application souhaitée.

4 Différentes solutions

Nous allons désormais regarder quelles sont les principales solutions d'unikernels à l'heure actuelle et les comparer entre elles.

4.1 Solutions étudiées

L'ensemble des solutions étudiées jusqu'à présent dans le cadre de ce travail se basent toutes sur le même hyperviseur : Xen. Il s'agit donc d'une référence essentielle, crédible et fiable dans le domaine de la virtualisation. Cependant un nombre important de solutions n'hésitent pas à réimplémenter certaines parties de Xen, comme le XenStore [3].

KylinX [2] offre un mécanisme de pVM, pour *process-like VM*. Ce procédé permet de réaliser des `fork` en instanciant une nouvelle machine virtuelle. La machine ayant fait l'appel à `fork` sera mise en pause le temps de la création de la pVM et recevra ensuite un moyen de la contacter.

LightVM [3], une nouvelle solution de virtualisation permet de démarrer une machine virtuelle presque aussi rapidement qu'un `fork` ou un `exec` sur Linux et serait deux fois plus rapide que Docker, une solution permettant de lancer des conteneurs.

Tinyx [3], une solution de *build* automatisée permet de créer des images de machines virtuelles Linux minimalistes en utilisant une approche particulièrement originale. En effet, il va tenter de retirer une à une les différentes options de boot en testant si l'application continue toujours de fonctionner comme souhaitée avec l'aide d'un jeu de test. Si un des tests ne passe plus, l'option est réactivée, car essentielle.

4.2 Solutions restantes à étudier

Je pense continuer, dans un premier temps jusqu'à mi-avril, à regarder d'autres solutions tout en approfondissant celles que j'ai pu trouver jusqu'à présent. Voici certaines des solutions que je souhaiterais étudier prochainement :

— Jitsu [4] qui utilise les unikernels pour servir des applications. Les auteurs ont fait quelques optimisations sur Xen, notamment pour le faire fonctionner sous ARM.
— OSv [5], en vérifiant si la solution ne serait pas trop axée pour la JVM, qui est la machine virtuelle permettant l'exécution de programmes Java.
— les exokernels [6], puisque ce serait de cela que s'inspireraient les unikernels. Dans la même mesure, des alternatives telles que les lambda d'Amazon par exemple [7,8] permettrait de voir ce qui se fait à côté.

Ensuite, d'avril jusqu'à mai, je compte regarder en détail la partie concernant l'évaluation des performances en comparant les différentes solutions entre elles. Enfin, je compte finaliser la rédaction du rapport final durant le mois de mai.

5 Évaluation des performances

Dans cette partie il faudra énumérer les limites des différentes solutions étudiées, et évaluer leurs performances, vérifier si les gains en performances n'impactent pas la sécurité, être critique sur les résultats présentés dans les divers papiers.

Une présentation idéale serait sous forme d'un tableau récapitulatif, offrant ainsi aux lecteurs une vision d'ensemble directe sans avoir à lire l'ensemble de ce rapport.

Ci-dessous un exemple de tableau :

Solution	Temps de boot
KylinX (pVM fork)	1.3 ms
LightVM	4 ms

6 Conclusion

Faire le bilan de l'ensemble, parler de l'avenir des unikernels, orientations dans le monde de la recherche et les différentes pistes à creuser.

Références

[1] A. Madhavapeddy, D.J. Scott, Unikernels: Rise of the Virtual Library Operating System, Queue 11(11) (2013) 30:30-44. 10.1145/2557963.2566628.

[2] Y. Zhang, J. Crowcroft, D. Li, C. Zhang, H. Li, Y. Wang, K. Yu, Y. Xiong, G. Chen, KylinX: A Dynamic Library Operating System for Simplified and Efficient

Cloud Virtualization, in: 2018 USENIX Annual Technical Conference (USENIX ATC 18), USENIX Association, Boston, MA, 2018, p. 173-86.

[3] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, F. Huici, My VM is Lighter (and Safer) Than Your Container, in: Proceedings of the 26th Symposium on Operating Systems Principles, ACM, New York, NY, USA, 2017, p. 218-33.

[4] A. Madhavapeddy, T. Leonard, M. Skjegstad, T. Gazagnaire, D. Sheets, D.J. Scott, R. Mortier, A. Chaudhry, B. Singh, J. Ludlam, others, Jitsu: Just-In-Time Summoning of Unikernels., in: NSDI, 2015, p. 559-73.

[5] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti, V. Zolotarov, OSvOptimizing the Operating System for Virtual Machines, in: 2014 USENIX Annual Technical Conference (USENIX ATC 14), USENIX Association, Philadelphia, PA, 2014, p. 61-72.

[6] D.R. Engler, M.F. Kaashoek, J. O'Toole Jr., Exokernel: An Operating System Architecture for Application-level Resource Management, in: Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, ACM, New York, NY, USA, 1995, p. 251-66.

[7] I. Król Michałand Psaras, NFaaS: Named Function As a Service, in: Proceedings of the 4th ACM Conference on Information-Centric Networking, ACM, New York, NY, USA, 2017, p. 134-44.

[8] J. Spillner, Snafu: Function-as-a-Service (FaaS) Runtime Design and Implementation, CoRR abs/1703.07562 (2017).