sQemu : vers un stockage virtuel scalable

Kevin Nguetchouang, Pierre Olivier, Alain Tchana

Université de Lyon 1, Université de Manchester, ENS Lyon
Laboratoire LIP - Site Monod - 46 Allée d'Italie
69007 Lyon - France
kevin.nguetchouang@ens-lyon.fr, pierre.olivier@manchester.ac.uk, alain.tchana@ens-lyon.fr

Résumé
Contrairement aux autres ressources telles que le CPU, la mémoire et le réseau, pour lesquelles la virtualisation est efficacement réalisée par accès direct, la virtualisation de disque est particulière. Dans cet article, nous apportons trois contributions. Notre première contribution est de montrer par des mesures expérimentales que les chaînes longues entraînent des problèmes de performance et d'évolutivité de l'empreinte mémoire dans un second temps. Notre deuxième contribution est l'extension à la fois du format Qcow2 et de son pilote dans Qemu pour répondre aux défis d'évolutivité identifiés. Notre troisième contribution est l'évaluation approfondie de notre prototype, appelé ***sQemu***, démontrant qu'il apporte des améliorations significatives des performances. Par exemple, il améliore le débit de RocksDB d'environ ~40% par rapport à Qemu de base sur une chaîne de snapshots de longueur 500.

Mots-clés : stockage, virtualisation, disque virtuel

1. Introduction

Selon son coût attractif, le cloud computing est de plus en plus utilisé par plusieurs communautés. La virtualisation est la technologie clé qui rend possible le cloud computing et donc son succès. Cependant, la virtualisation, et donc le cloud computing, se fait au prix d'un certain surcoût sur les performances des applications. Cette surcharge a été bien étudiée [2, 6, 8, 15, 19, 12, 1, 5]. Bien que cela concerne tous les types de ressources (CPU, RAM, réseau, disque), elles ne sont pas toutes affectées avec la même intensité. La figure 1 montre la dégradation des performances due à la virtualisation pour un large éventail de benchmarks, notamment Stream [7] (mémoire intensive), NPB [3] (processeur intensif), netperf [17] (réseau intensif), ainsi que la commande Linux `dd` (disque intensif, orienté débit) et fio [10] (disque intensif, orienté latence), lorsqu' ils s'exécutent dans AWS EC2 (type

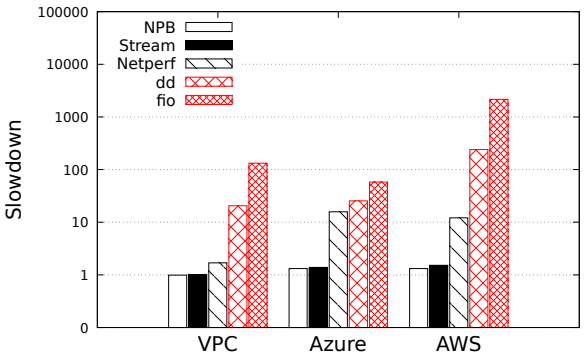


FIGURE 1 – Ralentissement des performances dû à la virtualisation pour différents types d'applications. Les résultats sont présentés en échelle log.

d'instance `t2.medium`), Microsoft Azure (type d'instance `Standard_B2s`), un cloud privé virtualisé et sur un cloud privé bare metal sans virtualisation [1]. Nous utilisons ce dernier comme base de référence. Nous pouvons observer que les deux applications gourmandes en disque (`dd` et `fio`) connaissent le ralentissement le plus élevé. Pour `fio`, c'est environ 1 639× la dégradation subie par NPB.

Étonnamment, contrairement aux autres types de ressources, très peu de travaux de recherche se concentrent sur l'amélioration de la virtualisation du stockage dans le cloud. Il est important de combler cette lacune, en particulier dans le contexte d'explosion de la vitesse des applications centrées sur les données (tendances big data, génomique, ML et IA). La virtualisation de disque est particulière car elle est toujours mise en œuvre via des architectures multicouches complexes [9, 20]. Une autre illustration de la singularité de la virtualisation de disque est le fait qu'elle se fait grâce à l'utilisation de formats de disques virtuels complexes (Qcow2, QED, FVD, VDI, VMDK, VHD, EBS, etc.) qui non seulement effectue la tâche de multiplexage du disque physique, mais doit également prendre en charge des fonctionnalités standard telles que les snapshots/restaurations, la compression et le chiffrement. Ces indirections sont à l'origine des surcoûts de virtualisation de disque.

Cet article se concentre sur Linux-KVM/Qemu (ci-après LKQ), une pile de virtualisation très populaire. LKQ prend en charge plusieurs formats de disques virtuels, parmi lesquels Qcow2 [16] est largement adopté en production [13]. Notre partenaire cloud, qui est un fournisseur de cloud public à grande échelle avec plusieurs centres de données répartis dans le monde, s'appuie sur LKQ et Qcow2. Une fonctionnalité importante fournie par Qcow2 est la capacité de créer des incrémentiels Copy-On-Write (COW) (fichiers de sauvegarde) afin de sauvegarder l'état du disque virtuel à un moment donné et de réduire l'utilisation de l'espace de stockage. Le disque virtuel d'une VM peut ainsi être vu comme une chaîne dépendant de plusieurs fichiers de sauvegarde. Dans cet article, nous identifions et résolvons les problèmes d'évolutivité de la virtualisation sur de telles chaînes de snapshots.

Le reste du papier est organisé comme suit. §2 présente l'arrière-plan. §3 et §4 présentent les problèmes d'évolutivité identifiés du format et notre solution pour y remédier. §5 présente les résultats d'évaluation de notre conception. §6 conclut l'article.

2. Linux-KVM/Qemu/Qcow2

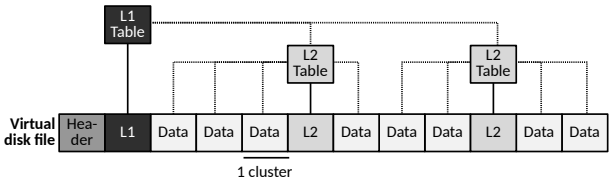


FIGURE 2 – Vue d'ensemble du format Qcow2.

Cette section présente le contexte nécessaire à la compréhension de nos contributions.

Présentation du format Qcow2

Le format Qcow2 permet de réaliser des snapshots en copie sur écriture en utilisant un mécanisme d'indexation mis en œuvre dans le format et géré au moment de l'exécution dans le pilote Qcow2, exécuté dans Qemu, pour faire correspondre les demandes d'E/S du processus utilisateur (en l'occurence la VM) adressant des secteurs/blocs virtuels aux décalages de l'hôte dans le(s) fichier(s) Qcow2. Un aperçu du format Qcow2 est donné à la Figure 2. Sans aucun snapshot, un disque virtuel est contenu dans un seul fichier. Le fichier est divisé en unités appelées clusters, qui peuvent contenir des métadonnées (par exemple, un en-tête, des tables d'indexation, etc.) ou des données qui re-

1. Nous avons choisi `t2.medium` et `Standard_B2s` pour correspondre à la taille de VM que nous avons utilisée dans notre cloud privé.

présentent des plages de secteurs consécutifs. La taille par défaut des clusters est de 64 Ko. L'indexation se fait par le biais d'une table à 2 niveaux, organisée comme un arbre radix : la table de premier niveau (L1) est petite et contiguë dans le fichier, tandis que la table de second niveau (L2) peut être répartie sur plusieurs clusters non contigus. L'en-tête occupe le cluster 0 au décalage 0 dans le fichier et les tables L1 viennent juste après l'en-tête. Pour des raisons de performances, les entrées L1 et L2 sont mises en cache en RAM.

Création de snapshots Qcow2

Un fichier F de disque virtuel Qcow2 peut être lié à un fichier de sauvegarde (snapshot), c'est-à-dire un fichier qui sera interrogé pour les clusters qui ne sont pas présents dans F. Aujourd'hui, la façon la plus courante de créer un snapshot incrémental en direct d'un disque virtuel F pour une VM donnée est de créer un nouveau fichier Qcow2 vide E et de le définir comme le disque actuel (appelé *volume actif*) pour la VM tandis que le disque virtuel précédent F est défini comme le fichier de sauvegarde pour E. De cette façon, toutes les opérations d'écriture effectuées par la VM seront dirigées vers le volume actif (E) tandis que les opérations de lecture seront dirigées soit vers E si les secteurs adressés y sont présents, soit vers les fichiers de sauvegarde dans le cas contraire. Avec le temps, les chaînes de fichiers de sauvegarde peuvent devenir très longues.

3. Problème des longue chaînes de snapshots

Une VM exécutée sur une longue chaîne de snapshots Qcow2 voit ses performances et son empreinte mémoire sérieusement affectées. Pour illustrer ces points, la figure 3 montre l'évolution de ces deux métriques pour une VM s'exécutant sur un disque virtuel avec des tailles de chaîne variables, allant de 0 à 300 snapshots. La taille totale du disque virtuel est de 20 Go et chaque snapshot contient une couche incrémentielle de 60 MO. Tous les fichiers résident localement sur le SSD de l'hôte. La VM a 4 Go de RAM allouée, 4 vCPU et fonctionne sous Ubuntu 18.04. Le débit de lecture est mesuré dans la VM en lisant l'intégralité du disque avec `dd` juste après 1) un premier appel à `dd` sur l'intégralité du disque pour s'assurer que les caches L1/L2 sont entièrement remplis et 2) un abandon du cache de page invité pour s'assurer que le fichier Qcow2 est accédé. L'empreinte mémoire est mesurée à partir de l'hôte comme la taille maximale de l'ensemble résident (RSS) de l'hyperviseur observée pendant l'exécution de la commande `dd`.

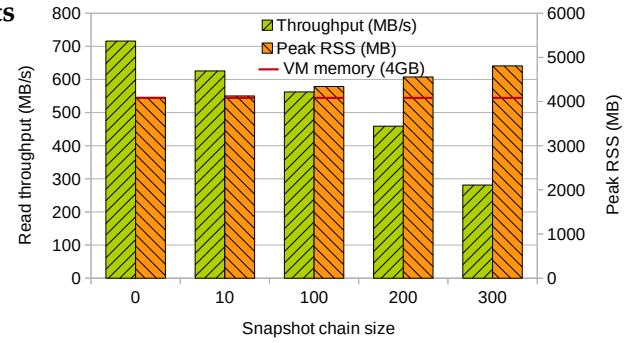


FIGURE 3 – L'évolution des performances d'E/S et de la mémoire en fonction de la taille de la chaîne

Comme on peut l'observer, bien qu'avec de petites chaînes le débit de lecture ne soit pas sensiblement affecté, lorsque la taille de la chaîne augmente, cette métrique chute de manière significative. Sur un disque virtuel avec une taille de chaîne de 300, le débit de lecture n'atteint que 39% de ce qui peut être obtenu sur un disque sans snapshots. En ce qui concerne la consommation de mémoire, avec peu ou pas de snapshots, la surcharge mémoire que Qemu présente en plus des 4 Go utilisés par la VM est négligeable. Cependant, avec de longues chaînes de snapshots, cette surcharge devient significative : avec 300 snapshots, 711 Mo de RAM supplémentaire sont consommés par Qemu. Troisièmement, nous avons utilisé le profileur de tas `massif` de Valgrind pour étudier la consommation de mémoire pendant le test `dd` sur les expériences

longues de 300 snapshots, et avons découvert que l'augmentation de l'empreinte mémoire est due à diverses structures de données qui sont allouées pour chaque snapshot dans la chaine au lieu d'avoir une structure allouée pour toute la chaine. La structure de donnée responsable ici le cache d'indexation L2 - y en a un par snapshot au lieu d'avoir un pour toute la chaine puisqu'une VM démarre sur une chaine de snapshots. Ces chiffres ont été recueillis sur Qemu 4.2 mais nous avons également confirmé ce comportement sur la dernière version (v6.0). Nous nous concentrons sur la version 4.2 dans la suite de cet article car c'est la version utilisée par notre partenaire fournisseur de cloud et la version par défaut installé dans Ubuntu.

4. sQemu : Scalable Qemu

Cette section présente une nouvelle version de Qemu et Qcow2 qui s'attaque aux deux problèmes d'évolutivité identifiés dans la section précédente, concernant les performances et la consommation de mémoire. Idéalement, les deux métriques devraient être aussi indépendantes que possible de la longueur de la chaîne de fichiers de sauvegarde.

4.1. Principes et défis

sQemu s'appuie sur deux principes clés, illustrés sur la figure 4 : 1) l'accès direct aux clusters d'indexation/données sur disque, quelle que soit leur position dans la chaîne, lors des demandes d'E/S des VMs ; 2) l'utilisation d'un seul cache d'indexation unifié, évitant la duplication des entrées du cache en étant indépendant de la longueur de la chaîne. Dans le reste du document, nous notons *vQemu* et *vQcow2* respectivement *vanilla Qemu* et *Qcow2* format actuel (pour dire version de base). Nous appliquons le premier principe par le biais d'une modification légère mais rétrocompatible de vQcow2, nécessitant le stockage de métadonnées supplémentaires dans les images de disque virtuel, ainsi qu'une mise à jour du pilote Qcow2 dans la pile de stockage Qemu : nous appelons cette évolution sQemu pour *scalable Qemu*. En attendant, l'application du second principe ne nécessite qu'une modification minutieuse du pilote Qcow2.

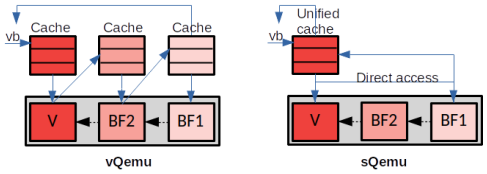


FIGURE 4 – vQemu comparé à notre conception sQemu, qui suit deux principes : accès direct et cache unifié.

L'un des principaux défis de l'implémentation d'sQemu concerne son intégration transparente et rapide dans l'infrastructure de notre partenaire de cloud computing (et dans les infrastructures de cloud computing en général). Notre solution doit tout d'abord être compatible avec les différents backends qui peuvent contenir des fichiers de sauvegarde sur disque dans l'infrastructure cloud d'aujourd'hui : ceux-ci peuvent être stockés directement sur le disque de l'hôte mais aussi accessibles par l'hôte via le réseau et servis par des serveurs NFS centralisés ou des systèmes de fichiers distribués. C'est pourquoi nous proposons de modifier un format de disque existant populaire plutôt que d'en proposer un nouveau [18]. Un défi connexe est également la rétrocompatibilité : les images Qcow2 existantes qui n'ont pas les métadonnées de notre format devraient toujours fonctionner avec notre version mise à jour de Qemu (sans gain de performance/ consommation de mémoire sur les longues chaînes), et les images utilisant notre format devraient également fonctionner avec la version de base de Qemu qui n'exécute pas notre pilote Qcow2 mis à jour (encore une fois sans gain sur les longues chaînes).

4.2. Amélioration du format

Lorsqu'un invité émet une requête d'E/S, vQemu scanne séquentiellement le volume actif et tous les fichiers de sauvegarde de la chaîne jusqu'à ce que le bon soit trouvé, ce qui n'est pas ef-

ficace. Nous proposons de mettre légèrement à jour le format Qcow2 ainsi que ses algorithmes de gestion dans Qemu afin d'éliminer cette opération de balayage de la chaîne. À cette fin, nous introduisons une nouvelle métadonnée dans le format indiquant, pour chaque cluster de données, le fichier de sauvegarde qui contient la dernière version (c'est-à-dire valide) du cluster. Nous appelons cette métadonnée le `backing_file_index`. Pour ce faire, nous exploitons les bits inutilisés des entrées de la table L2 ce qui permet de ne pas avoir de surplus de données sur le fichier.Nous utilisons 16 bits pour coder le `backing_file_index` dans chaque entrée L2.

4.3. Accès direct

Avec l'accès direct, nous maintenons un seul cache unifié pour l'ensemble du disque, indépendamment de la longueur de la chaîne de fichiers de sauvegarde. Notre cache a la même organisation que le cache de base de Qcow2 présenté dans la Section 2. Pour rappel, une entrée de cache correspond à une slice de la table L2. Comme indiqué dans la section précédente, dans SQEMU , une entrée L2 contient `backing_file_index` en plus des valeurs par défaut de vQcow2. Si la slice et l'entrée L2 existent toutes deux dans le cache et que le `backing_file_index` contenu dans l'entrée L2 correspond au volume actif, alors il y a une correspondance dans le cache et l'offset du cluster de données à lire se trouve dans l'entrée L2. Si `backing_file_index` ne correspond pas au volume actif, il s'agit d'un des fichiers de la chaine de sauvegarde. Cela signifie que l'invité demande un cluster de données qui n'existe pas encore sur le disque virtuel. Si la slice n'est pas encore présente dans le cache, il y a un manque de cache et la slice est soit récupérée du volume actif s'il existe, soit allouée s'il n'existe pas. Ces opérations sont similaires à celles de vQemu.

5. Evaluation

Méthodologie

Nous comparons systématiquement SQEMU et VQEMU . Nous évaluons plusieurs configurations en faisant varier trois paramètres la longueur de la chaîne (1-1000) ; la taille du disque virtuel (50GB, 150GB) ; ainsi que la taille du cache (de 30% à 100% de la taille du cache nécessaire pour contenir la totalité des entrées L2 pour indexer un disque complet, c'est-à-dire de $1,9$ Mo à $6,25$ Mo pour un disque de 50 Go, et de $5,6$ Mo à $18,75$ Mo pour 150 Go). Pour toutes les expériences, les clusters valides sont répartis uniformément sur les fichiers de sauvegarde de la chaîne du disque. Le disque virtuel est peuplé à 90% de données aléatoires pour les expériences avec des micro-benchmarks utilisant la commande Linux `dd`, et à 25% pour les expériences avec des macro-benchmarks utilisant le client RocksDB [4]. La version de SQEMU inclut un script de génération de chaînes hautement configurable.
Sauf indication contraire, la taille du cache L2 est fixée de manière à ce qu'il puisse contenir toutes les entrées L2 pour indexer le disque entier. Tous les résultats présentés dans cette section sont une valeur moyenne de 5 exécutions.

5.1. Temps de démarrage d'une VM

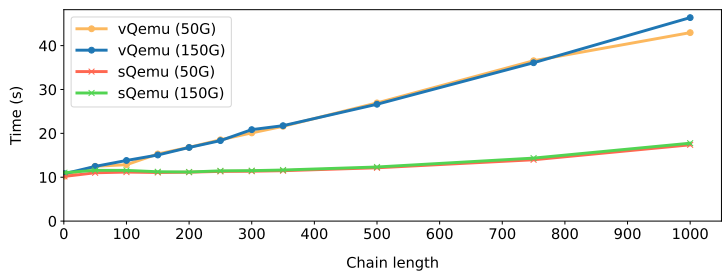


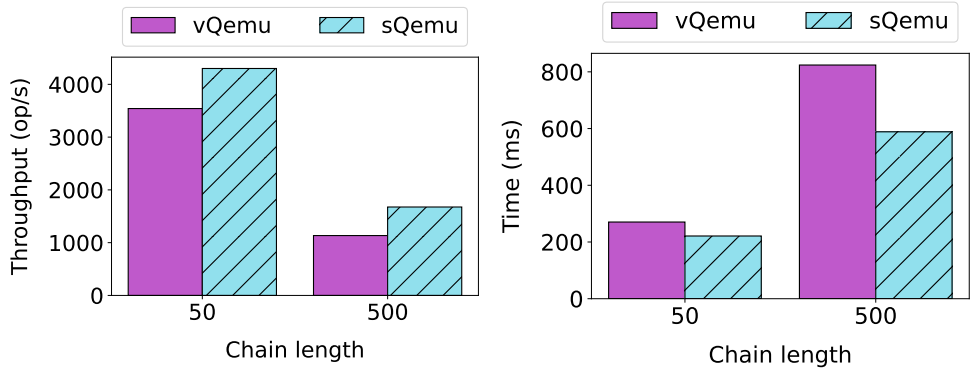
FIGURE 5 – Temps de démarrage de la VM.

Le temps de démarrage de la VM est une mesure critique dans le cloud [11, 14]. La figure 5 compare le temps nécessaire au démarrage d'une VM sous SQEMU et VQEMU en faisant varier la lon-

gueur de la chaîne et la taille du disque virtuel. Le temps de démarrage augmente rapidement avec la longueur de la chaîne sous VQEMU : il passe d'environ 10 secondes sur une chaîne de taille 1 à plus de 40 secondes (4×) sur une chaîne de taille 1000. Au contraire, avec SQEMU , cette augmentation est modérée : de 10 secondes à 17 secondes (1,7×).
Dans le premier cas, le surcoût est de 26 secondes, ce qui représente 250 %. Il augmente d'environ 70 % avec SQEMU (7*s*).

5.2. Workload réaliste : RocksDB-YCSB

Nous avons créé une base de données RocksDB qui remplit 40 % de la taille du disque de la VM, et l'avons alimentée à l'aide du client YCSB, générant une distribution uniforme de clusters valides dans les chaînes Qcow2 générées (chaines de 50 et de 500 snapshots pour ces expés). Nous utilisons YCSB-C, qui simule un utilisateur effectuant des requêtes en lecture seule (1 thread YCSB a été utilisé). Nous avons mesuré le débit et le temps d'exécution (les deux mesures de performance de RocksDB) de YCSB pour un total de 500K requêtes.



(a) Débit, avec une taille de cache de 3 Mo. **Haut c'est mieux.**

(b) Temps d'exécution, avec une taille de cache de 3 Mo. **Bas c'est mieux**

FIGURE 6 – RocksDB-YCSB résultats pour YCSB-C.

La figure 6a montre les résultats pour la mesure du débit. SQEMU est toujours plus performant que VQEMU pour les deux longueurs de chaîne (33% pour la longueur 50 et 47% pour la longueur 500).
La figure 6b présente les résultats du temps d'exécution. En ce qui concerne le débit, SQEMU améliore VQEMU . En considérant une chaîne de 50 snapshots, SQEMU réduit le temps d'exécution de YCSB de 22% pour une taille de cache de 3MB. Pour une chaîne de 500 snapshots, l'amélioration est d'environ 36 % avec une taille de cache de 3 Mo.

6. Conclusion

Nous présentons, pour la première fois, une analyse qui a révélé la présence de longues chaînes d'instantanés, ce qui entraîne des problèmes d'évolutivité tant au niveau des performances. Nous présentons SQEMU , une solution à ce problème, sous la forme d'une légère extension du format Qcow2 tout en préservant la rétrocompatibilité. Nous avons construit SQEMU en suivant les principes de l'accès direct et du cache à indexation unique, quelle que soit la longueur de la chaîne. Les évaluations ont montré que SQEMU améliore le débit d'E/S de RocksDB jusqu'à ~ 40% par rapport à VQEMU .

Bibliographie

1. Adams (K.) et Agesen (O.). – A comparison of software and hardware techniques for x86 virtualization. *ACM Sigplan Notices*, vol. 41, n11, 2006, pp. 2–13.
2. Barham (P.), Dragovic (B.), Fraser (K.), Hand (S.), Harris (T.), Ho (A.), Neugebauer (R.), Pratt (I.) et Warfield (A.). – Xen and the art of virtualization. *ACM SIGOPS operating systems review*, vol. 37, n5, 2003, pp. 164–177.
3. Division (N. A. S. N.). – Nas parallel benchmarks. https ://www.nas.nasa.gov/software/npb.html.
4. Facebook. – A persistent key-value store for fast storage environments, 2012. https ://rocksdb.org/.
5. Gordon (A.), Amit (N.), Har'El (N.), Ben-Yehuda (M.), Landau (A.), Schuster (A.) et Tsafrir (D.). – Eli : Bare-metal performance for i/o virtualization. *ACM SIGPLAN Notices*, vol. 47, n 4, 2012, pp. 411–422.
6. Huber (N.), von Quast (M.), Hauck (M.) et Kounev (S.). – Evaluating and modeling virtualization performance overhead for cloud environments. *CLOSER*, vol. 11, 2011, pp. 563–573.
7. John D. (M. C.). – Stream : Sustainable memory bandwidth in high performance computers. http ://www.cs.virginia.edu/stream/.
8. Lange (J. R.), Pedretti (K.), Dinda (P.), Bridges (P. G.), Bae (C.), Soltero (P.) et Merritt (A.). – Minimal-overhead virtualization of a large scale supercomputer. *ACM SIGPLAN Notices*, vol. 46, n7, 2011, pp. 169–180.
9. Le (D.), Huang (H.) et Wang (H.). – Understanding performance implications of nested file systems in a virtualized environment. – In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, *FAST'12*, FAST'12, p. 8, USA, 2012. USENIX Association.
10. Lofstead (J. F.), Klasky (S.), Schwan (K.), Podhorszki (N.) et Jin (C.). – Flexible io and integration for scientific codes through the adaptable io system (adios). – In *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, pp. 15–24, 2008.
11. Manco (F.), Lupu (C.), Schmidt (F.), Mendes (J.), Kuenzer (S.), Sati (S.), Yasukata (K.), Raiciu (C.) et Huici (F.). – My vm is lighter (and safer) than your container. – In *Proceedings of the 26th Symposium on Operating Systems Principles*, *SOSP '17*, SOSP '17, p. 218–233, New York, NY, USA, 2017. Association for Computing Machinery.
12. Morabito (R.), Kjällman (J.) et Komu (M.). – Hypervisors vs. lightweight virtualization : a performance comparison. – In *2015 IEEE International Conference on Cloud Engineering*, pp. 386–393. IEEE, 2015.
13. Moscovici (E.) et Abir (A.). – How to handle globally distributed qcow2 chains. – KVM Forum, 2017.
14. Nitu (V.), Olivier (P.), Tchana (A.), Chiba (D.), Barbalace (A.), Hagimont (D.) et Ravindran (B.). – Swift birth and quick death : Enabling fast parallel guest boot and destruction in the xen hypervisor. – In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, *VEE '17*, VEE '17, p. 1–14, New York, NY, USA, 2017. Association for Computing Machinery.
15. Padala (P.), Zhu (X.), Wang (Z.), Singhal (S.), Shin (K. G.) et al. – Performance evaluation of virtualization technologies for server consolidation. *HP Labs Tec. Report*, vol. 137, 2007.
16. Qemu Contributors. – Qcow2 documentation, 2020. https ://github.com/qemu/qemu/blob/master/docs/interop/qcow2.txt.
17. Rick (J.). – Hewlett-packard netperf benchmark. https ://github.com/HewlettPackard/netperf.

18. Tang (C.). – Fvd : A high-performance virtual machine image format for cloud. vol. 2, 2011.
19. Walters (J. P.), Chaudhary (V.), Cha (M.), Guercio (S.) et Gallo (S.). – A comparison of virtualization technologies for hpc. – In *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*, pp. 861–868. IEEE, 2008.
20. Zhou (R.), Sivathanu (S.), Kim (J.), Tsai (B.) et Li (T.). – An end-to-end analysis of file system features on sparse virtual disks. – In *Proceedings of the 28th ACM International Conference on Supercomputing*, *ICS '14*, ICS '14, p. 231–240, New York, NY, USA, 2014. Association for Computing Machinery.

Appendice

Environnement d'évaluation.

Pour avoir un environnement de test représentatif, nous utilisons 2 serveurs, l'un étant le nœud de calcul exécutant les VM, et l'autre le nœud de stockage contenant les fichiers des disques virtuels. Chaque serveur est équipé d'un processeur Intel Xeon Gold à 32 cœurs, cadencé à 2,10 GHz, de 192 Go de RAM et d'un SSD SATA Samsung MZ7KM480HMHQ0D3. Ils sont reliés par une connexion Ethernet 10Gbps. Le nœud de stockage sert les fichiers des disques virtuels via NFS. Les deux serveurs fonctionnent sous Debian 10 avec Linux 4.19.0 comme système d'exploitation hôte. Toutes les VM fonctionnent sous Ubuntu 18.04 avec Linux 4.15.0 et sont configurées avec 4 Go de mémoire et 4 vCPU. Sauf indication contraire, la taille du disque virtuel est de 50 Go.

Métriques et repères.

Nous recueillons deux types de métriques, *high-level* et *low-level*. Les premières sont celles qui ont un impact direct sur la qualité de service perçue par l'utilisateur final. Nous considérons le temps de démarrage de la VM, les frais de mémoire, le temps d'exécution de l'application et le débit des disques d'E/S. La surcharge mémoire est la mémoire supplémentaire consommée par Qemu en plus de la mémoire pseudo physique allouée à la VM. Les mesures de bas niveau représentent les coûts internes qui aident à expliquer les mesures de haut niveau. Il s'agit du nombre total d'échecs de cache, du nombre total de hits de cache non alloués et de la latence de recherche de cache. La latence de recherche est le temps nécessaire pour trouver l'offset valide d'un cluster de données dans le système de mise en cache. Pierre TODO : ceci n'est vrai que pour vanilla et ne vaut probablement pas la peine d'être mentionné ici. Les benchmarks de stockage sont exécutés dans les invités. Nous utilisons des microbenchmarks, dont Linux `dd` (qui lit séquentiellement le disque entier depuis l'invité, c'est-à-dire `dd if=/dev/sda of=/dev/null bs=4M`) et fio [10], ainsi que des macrobenchmarks, RocksDB-YCSB [4] et une mesure du temps de démarrage de la VM.