Etude comparative d'outils de virtualisation

13 novembre 2014

Introduction

Dans le domaine de virtualisation on peut distinguer deux catégories : ce qu'on appelle **"lightweight"** virtualisation (ex :**LXC**) et ce qu'on peut appeler **"heavyweight"** virtualisation.

En effet,lors d'un choix, plusieurs outils de virtualisation ,s'inscrivant dans les deux catégories, s'imposent, mais la question qui se pose, le quel allons nous choisir ? sur quels critères allons nous fonder notre jugement ?

Ce document a été conçu dans le but de répondre à ces questions, toutefois, il reste sujet d'amélioration.

1 LXC (Docker) VS KVM

Dans cette première partie, nous allons entamer une étude compratif entre les "Linux Containers"(lightweight) et les "Kernel based VM".

KVM , contrairement à **LXC Docker**, est un hyperviseur(*hypervisor*) qui fait tourner des OS (système d'expoitation) sans les modifier, il utilise les fonctionalités de la virtualisation du hardware afin reduire la compléxité et l'overhead, à titre d'exemple : l'émulation des entrées/sorties qui réduit l'overhed. Cette virtualisation nous garantit une isolation quasi-physique des VMs.

Quant à **LXC Docker**, il est basé sur les **"containers"**, il ne se limite pas à faire tourner l'OS, mais il peut aussi le modifier, dans le but d'avoir une isolation des process,

par exemple on ajoute un **"containers ID"** pour chaque process. Linux containers est basé sur le concepte des **kernel namespaces**, les **namespaces** peuvent être exploitées de differentes façons, la plus répandue convient à créer un **"container"** isolé qui n'a pas de visibilité ni d'accessibilité aux objets qui lui sont éxterieurs.

Lors de cette étude nous allons nous basée sur un travail réalisé par **IBM**[2], où nous allons mettre l'accent sur les **"Couldy performences"**. Les résultats des tests concernent 15 VMs simultanément en marche & **OpenStack**.

Testes	LXC Docker	KVM
Temps moyen de démarage	3.52s	5.78s
Temps moyen de redémarage	2.57s	124.43s
Temps moyen de suppression	3.56s	3.48
Temps moyen de Snapshot VM to Image	36.93s	48.02s
Temps moyen de suppression	3.56s	3.48s
Usage CPU temps total (sys)	0.17%	1.4%
Usage CPU temps total (usr)	0.54%	7.4%
Usage CPU etat stable (sys)	0.03%	0.36%
Usage CPU etat stable (usr)	0.2%	1.91%
Usage CPU démarage (sys)	0.57%	2.23%
Usage CPU démarage (usr)	1.39%	13.45%
Usage CPU redémarage (sys)	0.26%	0.18%
Usage CPU redémarage (usr)	0.69%	0.84%
Usage CPU Snapshot VM to Image (sys)	0.15%	1.0%
Usage CPU Snapshot VM to Image (usr)	0.42%	1.46%
Usage de mémoire état stable	49Mb/VM (734Mb)	292MB/VM (4387Mb)
Usage de mémoire au démarage	45Mb/VM (677Mb)	182MB/VM (2737Mb)
Usage de mémoire au redémarage	48Mb total	486Mb total
Usage de mémoire au Snapshot VM to Image	57Mb total	114Mb total
Calcule des nombres premier jusqu'a 20000	15.22s	15.13s
Network throughput (Mbits/s)	940.26	940.56

TABLE 1 – Cloudy Performence : **LXC Docker** vs **KVM** (OpenStack)

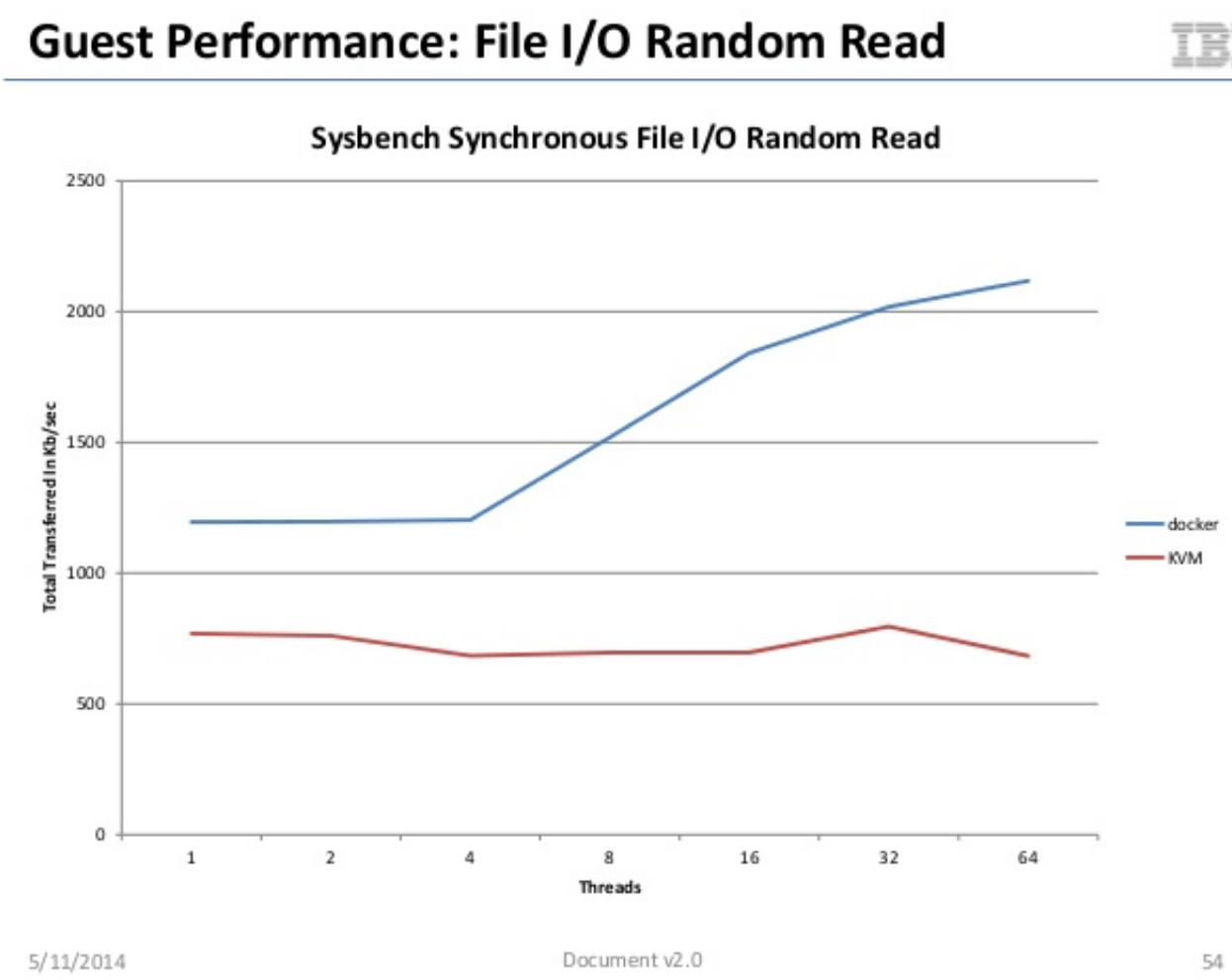

FIGURE 1 – Lecture aléatoire I/O

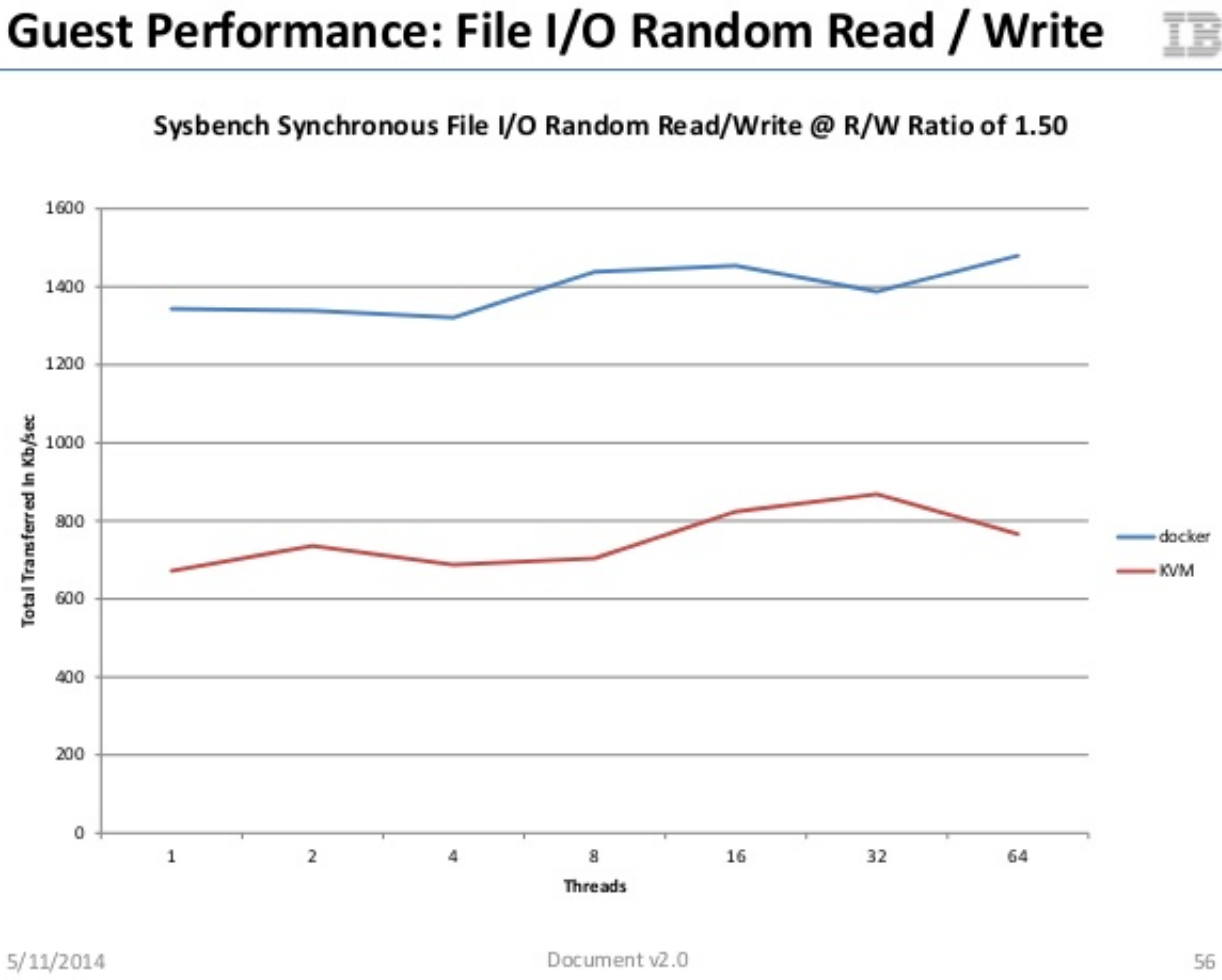

FIGURE 2 – Lecture/Ecriture aléatoire I/O

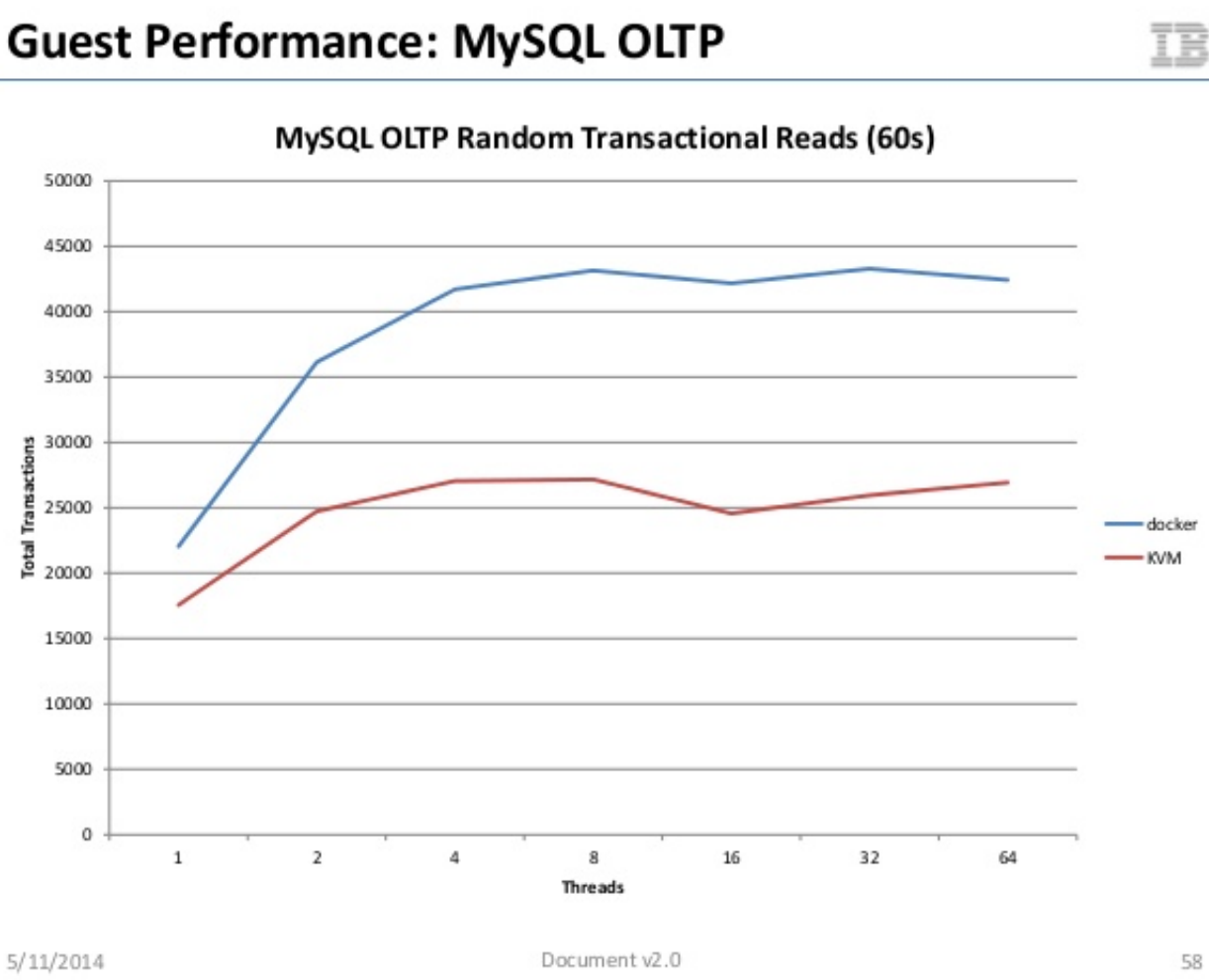

FIGURE 3 – Mysql : Lecture transactionnelle (60s)

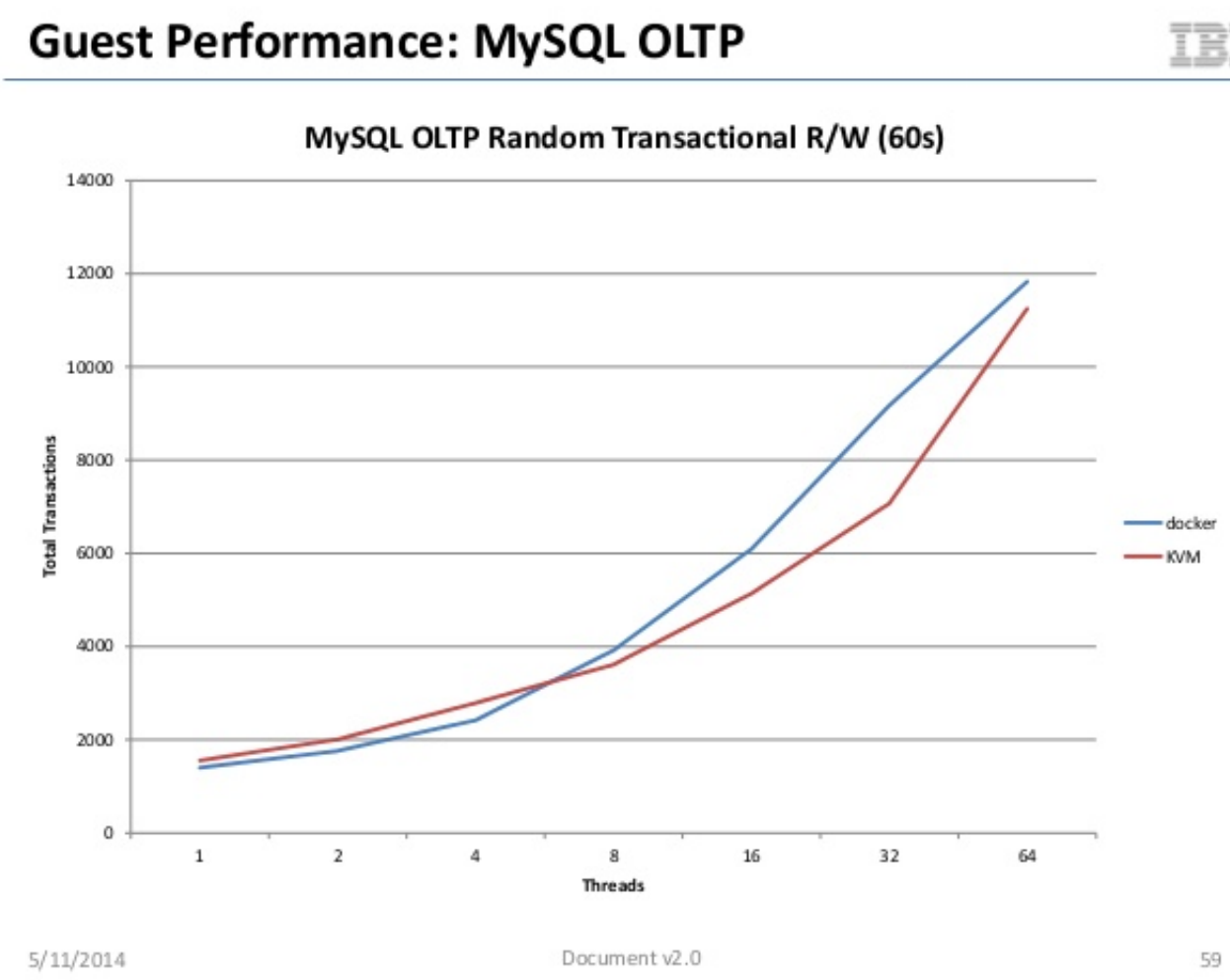

FIGURE 4 – Mysql : Lecture/Ecriture transactionnelle (60s)

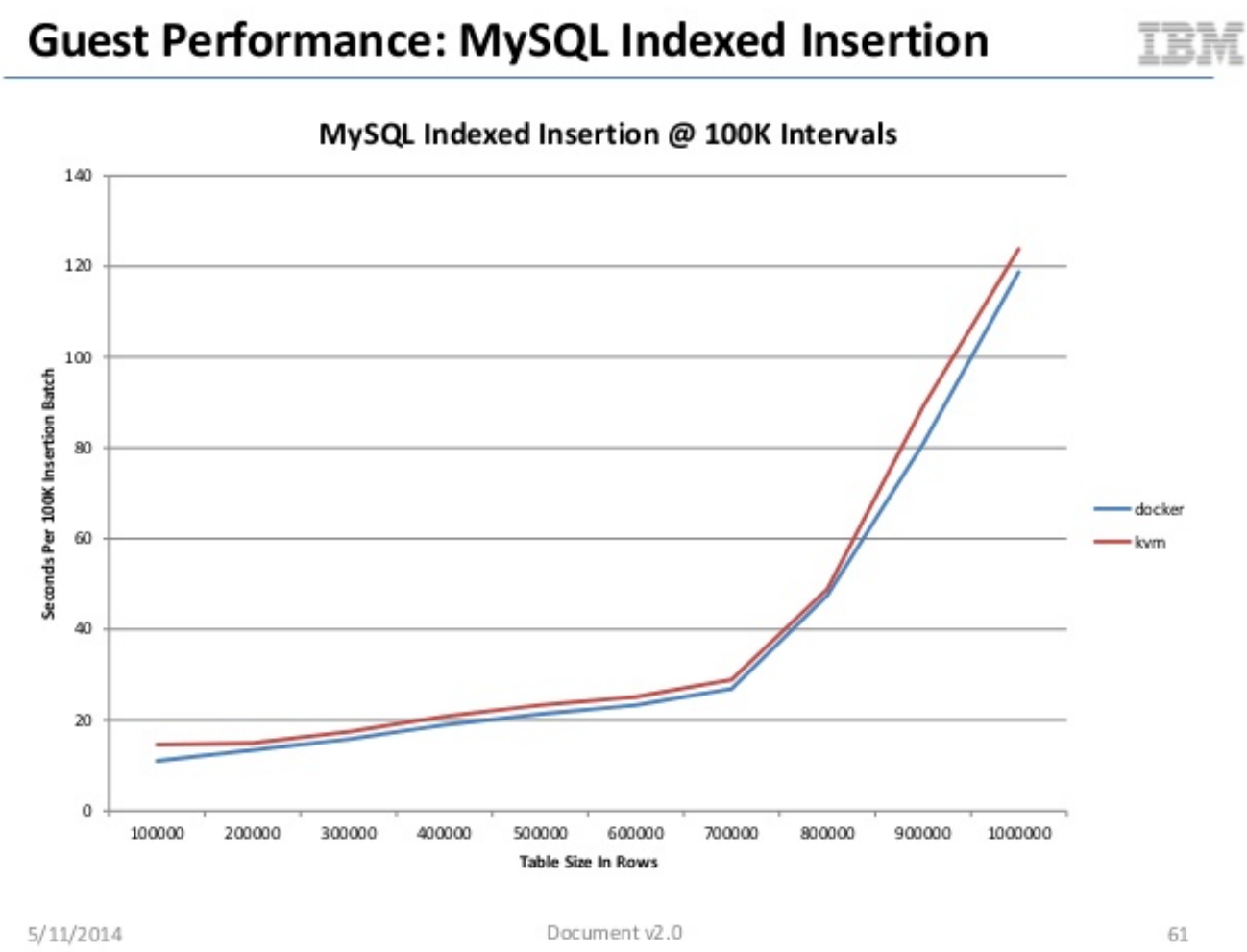

FIGURE 5 – Mysql : insertion indexée

Conlusion

En somme, on remarque qu'au niveau des performances ces outils sont quasi egaux, mais **LXC Docker** prend le dessus au niveau d'exploitation des ressources.

2 LXC Docker vs OpenVZ vs LMCTFY(Open source google linux containers) vs KVM

A la lumière de la première partie, nous allons etudier de façon rapide deux nouveaux outils et les comparer au deux premiers.

Openvz et **LMCTFY**, tout comme **Docker**, basés sur le principe des **"containers"**, sauf que **OpenVz** a besoin de copier et charger l'image en mémoire.

donc pour ne pas tarder nous allons passer aux resultats experimentaux obtenu par l'etude detaillée dans l'article [1].

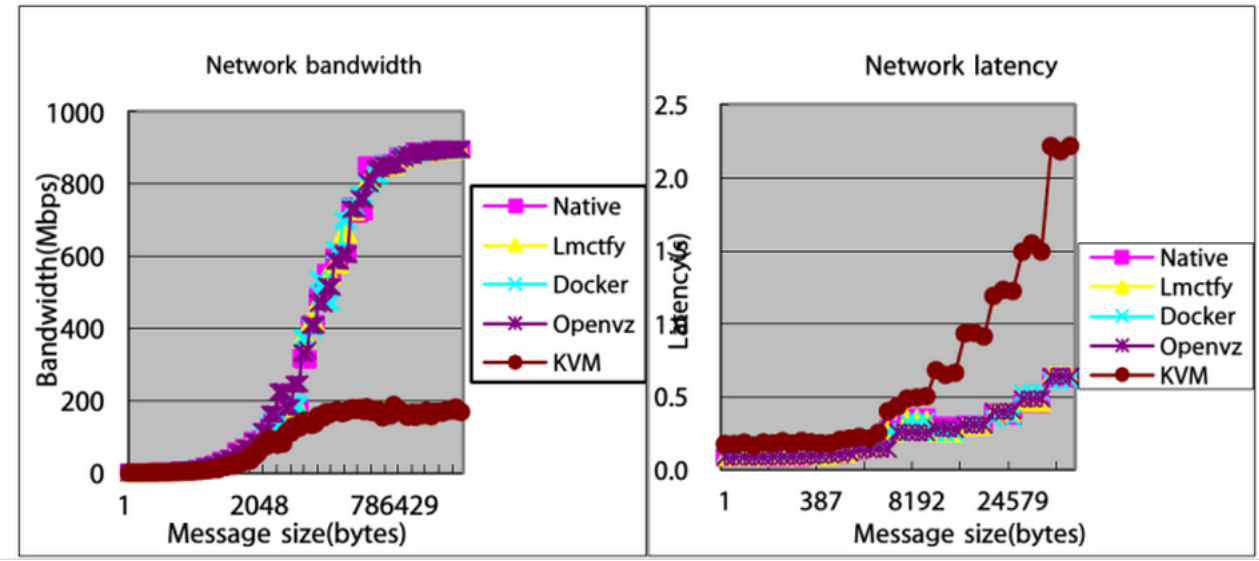

FIGURE 6 – Network Bande-passant/Latence

	Lmctfy	Docker	OpenVZ	KVM
CPU stress	0	0	0	0
Memory stress	53.3%	43.3%	20.1%	4.2%
Disk stress	34.4%	34.7%	36.8%	20.9%
Network Receivee	9.1%	6.7%	5.1%	3.0%
Network Sender	6.7%	6.3%	2.1%	0
Fork Bomb	DNR	DNR	0	0

FIGURE 7 – Performance sur test

les tests de stress realisés montrent que **CPU** n'est pas influencé par ces derniers, or que la memoire et le disque l'est, surtout pour les outils basé sur les **"containers"**, ceci est due au partage du même **OS kernel**, en effet lors du traitement d'un appelle d'un **Guest** stressé, il ne peut pas traiter les autres appelles venant d'autre **Guest** stressés.

Quant au test de **fork bombe**, **Docker** et **LMCTFY** lui sont vulnérables, car ils ne limites pas le nombre de prossecus fils créer par **Cgroups**, ceci représente un faille de securité.

	Lmctfy	Docker	OpenVZ	KVM
Time-consumings	0.202	0.189	13.197	21.588

FIGURE 8 – Vitesse de creation de VM

	Efficiency(%)	Isolation(%)	Speed(%)
Lmctfy	97.5	49.5	93.6
Docker	95.0	55.8	100
OpenVZ	88.5	85.3	1.5
KVM	65.0	94.7	0.9

FIGURE 9 – Performance globale

Conclusion

Pour conclure, **Docker** et **LMCTFY** ont une utilisation de ressource efficace ainsi qu'un très rapide deploiement mais la qualité de l'isolation laisse à désirer. **OpenVZ** moins efficace au niveau d'utilisation des ressource, offre une isolation assez bonne, mais trés lent au niveau du deploiement. **KVM** offre une isolation est très forte, mais une efficacité et rapidité médiocres.

Références

[1] X.Tang & Al. Performance evaluation of light-weight virtualization, 2009. 9

[2] Boden Russell of IBM. Passive benchmarking with docker, kvm & openstack, 2014. 2