Université de Reims Champagne Ardennes

Mise en place d'une solution d'évaluation des performances de solutions de virtualisation, basée sur une expérience utilisateur

Auteur :
HERARD Joffrey

Responsable :
Olivier FLAUZAC

29 avril 2017

Résumé

L'objet de notre étude a porté sur l'ensemble des solutions de virtualisation disponibles et possibles dans le cadre de la machine support mise à disposition. Il semble essentiel de rappeler que ces solutions sont variées dans leur fondamentalisme, leurs principes ou bien encore dans leurs domaines traditionnels d'utilisation. Afin de mettre en place une solution d'évaluation des solutions de virtualisation du point de vue des utilisateurs nous avons défini des scénarios de tests et des domaines d'études spécifiques de test sur chaque composant du système qui pourrait être utilisés par un ensemble d'utilisateurs. Pour cela, il a fallut définir ce qui allait être étudié sur chaque composant mais aussi comment mettre en oeuvre ces tests sur l'ensemble de ces machines. Je pus alors me poser un certain nombre de questions : Quels outils existent ? Comment faire la mise en place de ces outils ? Comment faire l'analyse de l'ensemble des résultats ? Comment faire l'établissement des liens et d'une échelle de performances ? Comment faire pour délimiter les meilleurs domaines de chaque solutions de virtualisation ?

Table des matières

Chapitre 1

Présentation du projet

Au travers de cette présentation il sera présenté l'ensemble du sujet, ainsi que les problématiques qu'il soulève, une définition précise des différents termes employés, l'explication des choix qui ont été fait sur certaines technologies et enfin l'ensemble des solutions émises pour résoudre les problématiques levées.

1.1 Sujet

Ce stage ayant pour sujet la mise en place d'une solution d'évaluation des performances de solutions de virtualisation basée sur une expérience utilisateur, il a tout d'abord été nécessaire de définir plusieurs termes afin de cerner ce dont-il était vraiment question. Ainsi, j'ai d'abord déterminé les termes qu'il était intéressant de développer, tels que : les solutions de virtualisation, l'évaluation de ces solutions et l'expérience utilisateur. Pour commencer, portons notre attention sur les solutions de virtualisation afin d'illustrer de quoi il s'agit.

Définition 1. *Solutions de virtualisation : La virtualisation consiste à faire fonctionner un ou plusieurs systèmes d'exploitation / applications comme un simple logiciel, sur un ou plusieurs ordinateurs. Actuellement séparé en deux grands domaines, l'hyper-vision (eux même divisés en deux types : Type 1 et type 2) et la conteneurisation.*

Après avoir défini en détails les termes des solutions de virtualisation il est nécessaire d'expliquer le processus d'évaluation des performances.

1.2 Objectifs

1. Savoir évaluer les différentes solutions de virtualisation.
2. Effectuer une démarche scientifique cohérente.
3. Effectuer un travail collaboratif avec des personnes d'expériences sur le sujet, ce qui malgré tout est plus ou moins inédit pour moi (J'entends que la plupart des travaux réalisés ces dernières années ont été fait avec des gens de même expérience que moi).
4. Établir des statistiques, sur les résultats qui seront obtenus. En ressortir des résultats des variables qui sortent du lot.
5. Établir un tableau récapitulatif afin de pouvoir résumer toutes les données obtenues.

1.3 Problématique soulevée

Comment mettre en oeuvre des scénarios afin d'évaluer les différentes solutions de virtualisation et de conteneurisation du point de vue utilisateurs ? Tout cela de manière efficace et juste sans interférer dans les résultats.

1.4 Hypothèse de solution

Il existe un ensemble de logiciels type pour le provisionning de machine virtuelle ainsi que des outils permettant de réaliser des benchmark neutres dans leurs prises de ressources.

Chapitre 2

Analyse de l'existant

Dans ce chapitre, nous verrons quelles technologies existent pour répondre à notre problématique.

2.1 Virtualisation

2.1.1 Hyperviseurs

Définition 2. *Hyperviseurs : C'est une plate-forme de virtualisation qui permet à plusieurs systèmes d'exploitation de travailler sur une même machine physique en même temps. Il en existe deux catégories :*

- *La première bien nommée Type 1 : est un logiciel de virtualisation installé directement sur le matériel informatique, il contrôle non seulement le matériel, mais aussi un ou plusieurs systèmes d'exploitation invités.*
- *La deuxième bien nommée Type 2 : sont des applications de virtualisation qui s'exécutent non pas directement sur du hardware mais sur un système d'exploitation.*

Il existe un ensemble d'Hyperviseur de type 1, que l'on peut lister de manière non exhaustive :

- CP
- XEN
- ESX Server
- LPAR
- Hyper-V
- Proxmox
- KVM

Il existe un ensemble d'Hyperviseur de type 2, que l'on peut lister de manière non exhaustive :

- VMWare Server
- VMWARE Workstation
- QEMU
- Hyper-V
- Parallels Workstation
- Parallels Dekstop
- VirtualBox

2.1.2 Conteneurs

Définition 3. *Conteneurs : C'est de créer des instances dans un espace isolé au lieu. En d'autres termes, le partage du matériel est de rendre disponible de nombreux opérateurs sur le noyau lui-même plutôt que d'une autre couche.*

Il existe un ensemble de gestion de conteneurs, que l'on peut lister de manière non exhaustive :

- LXC
- Docker

2.2 Provisionning

Définition 4. *Provisionning : C'est l'approvisionnement de machines, afin de mettre en place des configurations, des allocations automatiques de ressources, voir même des installations de logiciel, gestions de configuration, maintenance système. Globalement il sert à faire de la gestion de groupe de machines.*

Initialement le provisioning était du scripting manuel, voir même des solutions Client/Serveur, avec un serveur de configuration et un ensemble d'agents de gestion placés sur les machines à administrer. On appelle cela un framework d'exécution distantes depuis une station de gestion. Le scripting a malgré tout des limites, cela représente un travail fastidieux, il fait souvent face à un problème d'hétérogénéité. On peut se retrouver face un script spécifique pour une opération, par serveur, par service. Voici une liste :

— Ansible
— Vagrant
— Saltstack
— Libvirt

2.3 Benchmarking

Une évaluation des performances est appelée souvent dans le monde anglophone un Benchmark. Ce terme est défini comme suit.

Définition 5. *Benchmarking : Évaluation des performances d'un système par simulation des conditions réelles d'utilisations.*

Il existe un certain nombre de benchmark référence en majeur partie par OpenBenchmark.com, les sources de ces benchmark sont réalisées avec le célèbre outil nommé Phoronix tests suite.

Chapitre 3

Analyse des besoins

Dans cette troisième partie, il va être abordé l'ensemble des besoins qui permettront d'émettre un début de réponse par rapport au sujet et à la problématique.

3.1 Plan d'expériences

Après une analyse des besoins fonctionnels du projet, il va être détaillé chaque étapes du test.

3.1.1 Sur les tests

Lors de l'élaboration des tests qui seront exécutés il sera nécessaire de choisir précisément comment et grâce à quoi nous pourrons évaluer chaque composants. Rappelons également qu'il y aura un ensemble de scénarios à définir et qu'il faudra effectuer différentes variantes de tests pour chaque technologies de virtualisation, et de conteneurisation. Afin d'avoir des résultats cohérents, la neutralité de la sonde devra être importante. De plus, nous devrons définir la composante environnementale de chaque tests. Dès lors, on appellera "environnement" ce qui ne serait pas jugé comme influent à travers mes tests alors qu'il le sera. Prenons l'exemple de la vitesse du processeur, de sa température, ou encore de sa vitesse de connexion pour les tests réseaux. Bien entendu l'alimentation des composants peut varier et la politique de gestion des ressource vis à vis de chaque technologie de conteneurisation et d'hypervisions aussi. Ainsi, pour chaque composant la question sera toujours la même, à savoir :

Que testons nous et pourquoi ?

Pour chaque composants détaillés ci-dessous, un schéma est en annexes concernant le détail du cheminement de chaque tests possibles pour chaque éléments.

3.1.2 Expérimentation disque dur

3.1.2.1 Pourquoi tester le disque dur ?

Le disque dur est sans aucun doute le membre de la machine le plus important pour le stockage de données. Il peut-être utilisé lors de l'enregistrement de fichiers, pour de l'écriture pure, l'enregistrement d'évènements locaux ou bien encore pour télécharger, assez simplement, des fichiers. D'un point de vue des utilisateurs, c'est le support de leurs usages, de leurs stockages d'informations. Ainsi, un utilisateur peut décider d'y stocker un ensemble de petits fichiers, de «gros» fichiers, voir de Tera-nesque fichiers.

3.1.2.2 Tester le disque dur sur quoi ?

L'architecture de l'ordinateur à toute son importance pour les tests, notamment la taille des blocs pour l'écriture ou la lecture. Néanmoins, l'évaluation sur le test se fait-elle avec un accès en séquentiel ou en aléatoire ? De manière textuelle nous pouvons résumer cela ainsi :

— Test de Performance en Écriture

1. Évaluation des temps de réponses.
2. Évaluation des Vitesse de transfert.
3. Évaluation de la meilleure performance possible (Maximum en vitesse écriture) sur des fichiers de différentes tailles.

— Test de Performance en Lecture

1. Évaluation des temps de réponses.
2. Évaluation de la meilleure performance possible (Maximum en vitesse lecture) sur des fichiers de différentes tailles.

— Évaluation des vitesse du cache.

3.1.3 Expérimentation processeur

3.1.3.1 Pourquoi tester le processeur ?

Le processeur peut être sollicité lors de calculs, notamment dans le cadre des hautes performances.

3.1.3.2 Tester le processeur sur quoi ?

Un CPU peut être testé sur cette ensemble de choses-ci :
— Évaluation de la vitesse de la lecture Mémoire.
— Évaluation de la vitesse de la écriture Mémoire.
— Évaluation de la vitesse de la copie Mémoire.
— Évaluation de la vitesse de calcul flottant à précision simple.
— Évaluation de la vitesse de calcul flottant à précision double.
— Évaluation des opérations d'entrée sortie par secondes.
— Évaluation des opérations de chiffrement.
— Évaluation des opérations de hachage.

3.1.4 Expérimentation carte graphique

3.1.4.1 Pourquoi tester la carte graphique ?

3.1.4.2 Tester la carte graphique sur quoi ?

— Évaluation de la vitesse de la lecture Mémoire.
— Évaluation de la vitesse de la écriture Mémoire.
— Évaluation de la vitesse de la copie Mémoire.
— Évaluation de la vitesse de calcul flottant à précision simple.
— Évaluation de la vitesse de calcul flottant à précision double.
— Évaluation des opérations d'entrée sortie par secondes.
— Évaluation des opérations de chiffrement.
— Évaluation des opérations de hachage.

3.1.5 Expérimentation réseaux

3.1.5.1 Pourquoi tester le réseaux ?

3.1.5.2 Tester le réseaux sur quoi ?

— Chaque tests doit être réalisés avec des paquets de taille croissante avec le temps.
— Vitesse de téléchargement à estimer.
— Vitesse d'envoi à estimer.

3.2 Choix sur les outils de virtualisation

Au vue des différentes définitions apportées précédemment il s'est avéré nécessaire de répartir nos tests sur l'ensemble des trois techonologies qui nous étaient accessibles afin de rester équitable dans notre démarche. Par conséquent, il faudra être cohérent dans nos choix et cela sans oublier que beaucoup ont, par effet de mode, abusé de l'utilisation des conteneurs. Nous avons alors décider d'étudier les outils Docker et LXC, assez populaires aujourd'hui. Pour commencer, il est nécessaire d'évaluer si Docker et LXC ont effectivement tous les points positifs qu'on leur vente. Docker étant basé sur LXC il serait intéressant d'évaluer si cette technologie, avec la couche apportée par Docker, a un vrai plus ou si cela est plutôt un moins. On a donc choisi d'évaluer la technologie de conteneurisation proposée par le conteneur Linux LXC et celle proposée par Docker. Parallèlement, nous pouvons partir du même constat pour les Hyperviseurs. Rappelons, qu'il en existe deux catégories : les types 1 & 2 et il faudra donc établir des tests sur chacun d'eux. Tout comme nous l'avions fait pour Docker et LXC nous pouvons commencer par nous inéresser à leur popularité respective. Ainsi, il serait intéressant de prendre le logiciel d'Oracle VirtualBox qui est hyperviseur de type 2 utilisant des technologie comme celles de QEMU, hyperviseur du même type. Il reste tout de même à évaluer KVM, QEMU, Hyper-V et VMWare. Si nous le

pouvions, il serait intéressant de comparer VMWare et Hyper-V car ils sont tous deux des acteurs importants dans les hyperviseurs d'aujourd'hui. De plus nous pourrions mettre en face à face QEMU, un hyperviseur de type 2 et VirtualBox. KVM est quant à lui un hyperviseur de type 1 intégré à Linux et utilisé dans Proxmox VE. En revanche, il semble essentiel de souligner que l'ensemble de nos évaluations dépendront des licences qui seront mises à notre disposition. Pour conclure, il est indispensable de savoir discerner ce qui est pertinent de ce qui ne l'est pas et cela sans omettre de différencier les hyperviseurs étudiés. Je suis donc revenu sur mon précédent choix en admettant plus judicieux d'utiliser la technologie propre plutôt qu'une surcouche (comme proposé par M. Flauzac au cours de nos échanges). En effet, un test est intéressant sur la technologie mais si on agrémente les technologies du surcouche, cela en reste t-il tout aussi pertinent ? Par conséquent le choix s'oriente sur :

— LXC
— Docker
— KVM
— QEMU
— Hyper-V
— VMWARE

Table 3.1 – Solutions de virtualisations

Conteneur	Hyperviseur
LXC	KVM
Docker	QEMU
	Hyper-V
	VMWARE

3.3 Choix d'outils d'évaluation

Dans cette section nous étudierons les outils appelés "sonde" dans les paragraphes précédents. Pour commencer nous expliquerons en quoi ces outils permettent la répartition de tâches lorsqu'on souhaite exécuter une évaluation et nous parlerons ensuite de la récupération de ces données et de ce qui va être prélevé.

3.3.1 Outils d'évaluation personnel

Après avoir observé les technologies de Provisionning et de test qu'il existait, nous nous sommes demandé s'il était pertinent d'avoir à développer un outil d'évaluations. Au vue de l'outil Phoronix, on s'est rendu compte que ce développement était compliqué et que nous n'aurions pas assez rapidement des résultats à exploiter.

3.3.2 Phoronix

Il était alors nécessaire de choisir un outil d'évaluations conséquent. L'outil Phoronix est un outil de suite de tests réputé et il n'est plus à sa version d'essai. Historiquement, le 5 Juin 2008, la version 1.0 du Phoronix Test Suite était annoncée et il est aujourd'hui à sa septième version, plus précisement la 7.0.1. Ce logiciel a été publié sous licence GPLv3 et est conçu pour permettre aux utilisateurs de partager, leurs tests de logiciels et de matériel informatique via une interface graphique.

3.4 Choix d'outils d'orchestration

Lors de ce stage il fallait pouvoir gérer un ensemble, suffisamment grand, de machines pour rendre le travail de scripting fastidieux. De plus, nous devions être capable de gérer le déploiement des machines virtuelles (KVM/QEMU et LXC/Docker) et provoquer l'exécution d'un ensemble de tests avant de les rapatrier sur les machines dites "maître". Nous avons alors utilisé quelques outils tels que Saltstack, Libvirt, Phoromatic.

3.4.1 Saltstack

L'outil Saltstack fonctionne avec une architecture Client/Serveur utilisant la technologie d'une file de messages ZeroMQ. Nous pouvons noter qu'il existe une autre technologie bien connue, celle de RabbitMQ. Vis à vis de son modèle Client/Serveur, Saltstack est dépendant d'une écoute de port et il a donc fallut envisager la possibilité de ports bloqués. Parallèlement, on appelle le serveur principal, qui émet les ordres, un "salt master" et les machines qui subissent les ordres des "salt minion".

3.4.1.1 Configuration Maître

Si on souhaite modifier un ensemble de choses tels que le réseaux, les ressources, la sécurité, les modules, l'architectures des fichiers, et les logs éventuels il sera nécessaire de modifier, sur le serveur les fichiers correspondants à la location /etc/salt/master. Dans le cas du réseau on pourra y paramétrer les interfaces, les ports d'écoute et les timeouts. Pour les ressources on peut configurer un certain nombre de fichiers ouverts, un certain nombre de travaux, le cache, etc...

3.4.1.2 Configuration Minion

Il est nécessaire sur le client, de modifier les fichiers correspondants à la location /etc/salt/minion, on doit donc y paramétrer le nom du serveur, et son adresse. On peut y gérer, comme sur le master, un ensemble d'architecture de fichier, une gestion des modules, la sécurité et les logs.

3.4.1.3 Détails sur le fonctionnement général

Un minion/client doit s'enregistrer auprès du serveur/master pour pouvoir en subir les ordres. Toute fois le serveur doit être en mode "acceptation". Une fois les machines clientes acceptées sur le serveur, l'ensemble des opérations se déroulent avec la commande "salt" en désignant : les machines cibles, une commande, une fonction personnelle et un module à exécuter.

3.4.2 Phoromatic

Le second outil Phoromatic a été proposé par la suite Phoronix, comme nous pouvons le deviner. C'est un serveur d'orchestration de lancement de tests réalisé en PHP et utilisant les web-sockets. Il nécessite un lancement sur la machine que l'on souhaite et requiert que chaque machines, visant à être évaluées, aient la suite de test Phoronix préalablement installée. On peut ensuite se connecter au serveur, grâce à une ligne de commande particulière, en lui spécifiant l'adresse, le port et le token associé. Cet outil nous permet alors de choisir la programmation des tests, autrement dit le support, la machine, le composant et l'heure que l'on souhaite.

3.4.3 Libvirt

Libvirt est à lui seul un ensemble d'outils utiles dans la gestion de machines virtuelles et lorsque l'on souhaite exécuter une action sur celles-ci. Il est capable d'effectuer une gestion : du stockage, du réseau, d'une installation, de clonage de sauvegarde... L'Hôte des machines virtuelles, autrement dit la machine physique, y est alors appelé un "noeud". L'hyperviseur étant une couche logicielle de virtualisation avec des propriétés génériques et spécifiques, il s'exécute sur un "noeud". On note que Libvirt est capable de gérer un ensemble d'hyperviseur tel que :

- XEN
- Qemu/KVM
- LXC
- OpenVz
- VirtualBox
- VMware ESX Workstation Player
- Hyper-V
- IBM PowerVM

— Virtuozzo
— Bhyve

Le principe de Libvirt est de réaliser des opérations sur les domaines. N'ayant besoin que d'un hyperviseur et d'une adresse, la connexion à l'hyperviseur se fait par URL. L'exécution des commandes se fait en shell spécifique ou en commande shell paramétré avec un fichier XML. Libvirt est fondé sur une API écrite en : C, C++, C#, Java, PHP, Python, Ruby.

3.5 Impact des services de virtualisation sur les performances et donc les résultats

3.5.1 Impact d'un Hyperviseur

Le monde des Hyperviseurs étant assez vaste, chacun à une politique différente ne serait-ce que par le choix du type d'hyperviseur (Hyperviseur de type 1 & 2). Par conséquent, il y a forcément des Hyperviseurs qui vont avoir une politique différente sur l'accès concurrent à des ressources. Ils ont besoin d'isoler les interruptions et les accès à la mémoire et cela est très coûteux au niveau des performances. Les surcoûts en termes de performances pour visualiser un système comportent trois aspects principaux : la virtualisation du processeur, de la mémoire, et des entrées/sorties.

Processeur L'utilisation d'un hyperviseur au-dessous du système d'exploitation diffère du schéma habituel où le système d'exploitation est l'entité la plus privilégiée dans le système. De nombreuses architectures de processeur ne fournissent que deux niveaux de privilèges. Dans une virtualisation efficace l'utilisation du processeur à des implications critiques sur les performances des autres caractéristiques du système. Dans l'évaluation des performances d'une machine virtuelle, les processus sont gérés par la machine virtuelle à la place du système d'exploitation sous-jacent. Les threads émulés des environnements multi-thread, en dehors des capacités du système d'exploitation d'origine, sont gérés dans l'espace utilisateur à la place de l'espace noyau, permettant le travail avec des environnements sans support natif des threads. De bonnes performances sur un microprocesseur multi-cœur sont obtenues grâce à l'implémentation de threads natifs pouvant attribuer automatiquement le travail à plusieurs processeurs, ils permettent un démarrage plus rapide de processus sur certaines machines virtuelles.

3.5.2 Impact d'un conteneur

A la base, le concept de conteneurisation permet aux instances virtuelles de partager un système d'exploitation hôte unique, avec ses fichiers binaires, bibliothèques ou pilotes. Cette approche réduit le gaspillage des ressources car chaque conteneurs ne renferme que l'application et les fichiers binaires ou bibliothèques associés. On utilise donc le même système d'exploitation (OS) hôte pour plusieurs conteneurs, au lieu d'installer un OS (et d'en acheter la licence) pour chaque VM invitée. Le conteneur de chaque application étant libéré de la charge d'un OS, il est nettement plus petit, plus facile à migrer ou à télécharger, plus rapide à sauvegarder ou à restaurer. Enfin, il exige moins de mémoire. La conteneurisation permet au serveur d'héberger potentiellement beaucoup plus de conteneurs que s'il s'agissait de machines virtuelles. La différence en termes d'occupation peut être considérable, car un serveur donné accueillera de 10 à 100 fois plus d'instances de conteneur que d'instances d'application sur VM.

Chapitre 4

Prise de positions

Dans cette quatrième partie, nous allons expliquer le choix de certaines prises de positions pour les hyperviseurs, l'environnement, les tests et les différents scénarios.

4.1 Postulats

Pour commencer, nous allons développer l'ensemble des postulats.

4.1.1 Les hyperviseurs

Comme nous l'avons mentionné auparavant, il existe plusieurs types d'hyperviseurs : les hyperviseurs de type un et deux, et les conteneurs. Bien que nous soyons limités dans le temps il était tout de même important de tester au moins un hyperviseur de chaque type. Dès lors, les licences nécessaires à la mise en place d'un hyperviseur comme Hyper-V, VMWare, ORACLE VM Server n'étant pas libres et gratuites comme d'autres, nous avons faire un choix. Nous avons donc utilisé KVM, un hyperviseur de type 1, et QEMU, un hyperviseur de type 2 souvent associé à notre premier choix. Dans le cas des conteneurs, nous avons choisi LXC et Docker, abordés plus haut. Pour conclure, voici un tableau récapitulatif :

LXC ✓	Docker ✓	HyperV ≈
VMWare ≈	VirtualBox ✗	Qemu ✓
Oracle VM Server ✗	Proxmox ✗	KVM ✓
Hyper-V Containers ≈		

Comme nous pouvons le remarquer sur le tableau ci-dessus, nous avons quelques incertitudes concernant les Hyper-V de Microsoft et l'hyperviseur VMWare. Il y a deux raisons à cela, la première étant qu'il faudrait pouvoir passer du temps sur chaque technologies de virtualisations pour que les tests puissent être rigoureux et la seconde demeurant sur le fait qu'il est signalé qu'inclure la même procédure de test sur ces hyperviseurs serait intéressant.

4.1.2 L'environnement

Lorsque l'on fait des évaluations très précises il y a toujours un problème récurrent lié à ce que l'on appelle "l'environnement de test". Afin d'expliquer cela simplement nous pouvons illustrer notre idée de départ : Si on réalise un test sur une machine et qu'on lui donne 16 GB de RAM avec une fréquence X, on lui fait subire un changement matériel tel que, par exemple, une réduction de sa fréquence. La situation serait la même sur une machine équipée d'une même RAM mais pas à la même cadence. De part cet exemple il est évident que la réalisation de test, sur un état de cette machine puis sur un autre, ne serait pas pertinente pour les résultats. Je vais donc ici considérer des faits qui peuvent s'y apparenter mais cela de manière plus précise.

Postulat 1. *Chaque machine virtuelle, qu'elle soit engendrée par un Hyperviseur ou un service de Conteneurisation, sera une machine Debian.*

Ici ce choix, fut diriger simplement pour le côté libre de la distribution. La version de celle-ci est détaillée dans un autre postulat.

Postulat 2. *L'ensemble des machines virtuelles Debian qui seront clonées seront considérées identiques.*

Effectivement, les scripts utilisés sont, dans la plupart des cas, du clonage de machines virtuelle existantes, avec pour différence désirée, l'adresse mac. La pertinence des résultats ne peut être que renforcée par ce postulat. Aucune preuve n'existe pour prouver la véracité de ce postulat.

Postulat 3. *Chaque test sera réalisé en interne de chaque machine virtualisée et conteneurisée.*

Ce postulat répond à une demande du sujet "expérience utilisateur".

Postulat 4. *Tous les tests seront réalisés par la suite de test de Phoronix à la version 7.0.1 .*

Il est nécessaire de préciser cette prise de positions. Un test fait avec une autre version de phoronix engendrerait une différence de résultats.

Postulat 5. *La version de la Debian utilisée sera une version 8.7.1 .*

4.1.3 Nombres de machines

Il est nécessaire de voir l'impact des machines virtuelles entre elles et donc l'impact de la logique de l'hyperviseur sur les performances quand un nombre de machines virtuelles suffisamment grand est mis en jeu. Pour cela, nous réaliserons des tests dont le nombre de machines virtuelles sera un multiple de 5. Plus la difficulté se fera sentir sur les machines, plus le nombre sera malléable et sujet à changement. En effet, si vingt-cinq machines Docker fonctionnent correctement alors que trente machines sont trop lentes, nous pouvons utiliser la technique de séparation suivante : On prend les tests sur vingt-sept et, si les résultats sont acceptables, on augmente petit à petit le nombre de machine. Dans le cas contraires, on le réduira.

4.1.4 Sur les tests graphiques

La machine qui sert de support à l'ensemble des tests ne possédant par une carte graphique suffisamment puissante pour être pertinent dans les données récoltées sur plusieurs machines, nous avons lancé un test sur la machine physique afin d'évaluer les performances possibles. Cependant, celui-ci a duré trop longtemps pour avoir un aboutissement sur un ensemble de machines fini. De plus, lors de l'installation de la machine, aucun serveur X (Pour rappel la machine physique/ hôte est une Debian 8 version serveur) n'a été installé et ceci compromet la plupart des éventuels tests mis à notre disposition par Phoronix test suite. Mis cela a part, il est indispensable de procéder ce genre de test pour pouvoir en faire des scenarios.

4.1.5 Sur les tests reseaux

TO DO SCAPY

4.2 Les différents scénarios de tests

Au sein de cette section nous mettrons en avant les différents scénarios applicables dans notre univers de test. Ainsi, il sera mis en avant le scénario du Web sur les calculs processeurs, le disque et l'utilisation basique de réseaux. Enfin nous préciserons le nom des tests de chaque scénario et leur ordre.

4.2.1 Scénario 1 : Web

Ce premier scénario vise à évaluer la capacité de la machine virtuelle à gérer des cas de figure assez extrêmes tel qu'un nombre de requêtes élevé. Cela est typique du premier test d'Apache Benchmark qui a fait subir un million de requêtes en séquentiel à la machine. De plus, on le retrouve lors de l'envoi de mille requêtes en simultané jusqu'au nombre de 1 million, total cumulé, dans notre deuxième test PHP Benchmark. Voici donc la version des tests pour le web :

— pts/apache-1.6.1
— pts/phpbench-1.1.0

4.2.2 Scénario 2 : Calcul sur processeurs

Ce second scénario vise à évaluer la capacité de la machine virtuelle à gérer des cas de figure où le processeur est fortement sollicité. C'est le cas précisément lors d'une compression à l'aide de 7-zip, dans deux algorithmes d'intelligence artificielle nommés Crafty et TSCP ou bien encore lors de l'utilisation de PovRay, un logiciel assez gourmand en capacités.

— pts/7zip-compression-1.6.2
— pts/crafty-1.3.1

— pts/tscp-1.2.1
— pts/povray-1.1.3

4.2.3 Scénario 3 : Utilisations du disque

Dans ce troisième scénario nous avons plusieurs cas de figure ayant des obligations d'architecture, notamment pour la taille des blocs d'écriture et de lecture ainsi que pour la méthode d'accès, séquentielle ou aléatoire.
— pts/compilebench-1.0.0

4.2.4 Scénario 4 : Utilisation de la mémoire vive

La mémoire vive demeure une ressource enviée de tous, pour la mise en mémoire d'un programme un peu gourmand par exemple, mais c'est aussi un lieu de problème. En effet, il y a souvent des problèmes d'accès mémoire, en particulier lorsque plusieurs acteurs l'utilisent. Rappelons également que c'est une ressource où la vitesse est importante.
— pts/ramspeed-1.4.0 Type Copy - Benchmark integer
— pts/stream-1.2.0 Type Copy
— pts/t-test1-1.0.0 Thread 1

4.2.5 Scénario 5 : Utilisation Réseaux

A travers ce cinquième scénario il a été choisi d'évaluer la connexion sur la loopback afin de palier d'éventuels problèmes tels qu'un manque de temps pour la réalisation d'autres tests par exemple. Dès lors, des envoies en anneaux entre chaque machines virtuelles ou, mieux encore, un connexion en clique viendraient parfaire et affiner ce test.
— pts/network-loopback-1.0.1

Chapitre 5

Résultats

5.1 Partie Processeurs

5.2 Partie Disque dur

5.3 Partie Mémoire

5.4 Partie Réseaux

5.5 Partie Web

Chapitre 6

Bilan

Intro / Rappel Contexte
Nous avons donc pu en tirer la problématique suivante :

Problématique du sujet

Bla

6.1 Annexes

Annexes

Annexe 1

Intro

1 Partie 1

Bla

1.1 Sous-partie 1

Bla

1.2 Sous-partie 2

Bla

1.3 Sous-partie 3

Bla

2 Partie 2

Bla

2.1 Sous-partie 1

Bla

2.2 Sous-partie 2

Bla

2.3 Sous-partie 3

Bla

Annexe 2

Intro

Prérequis

Bla
— item1 ;
— item2 ;
— item3 ;
— item4.
Bla

1 Partie 1

Bla

1.1 Sous-parie 1

Bla

1.2 Sous-parie 2

Bla

2 Partie 2

Attention !
Texte d'avertissement

Bla

3 Partie 3

Bla

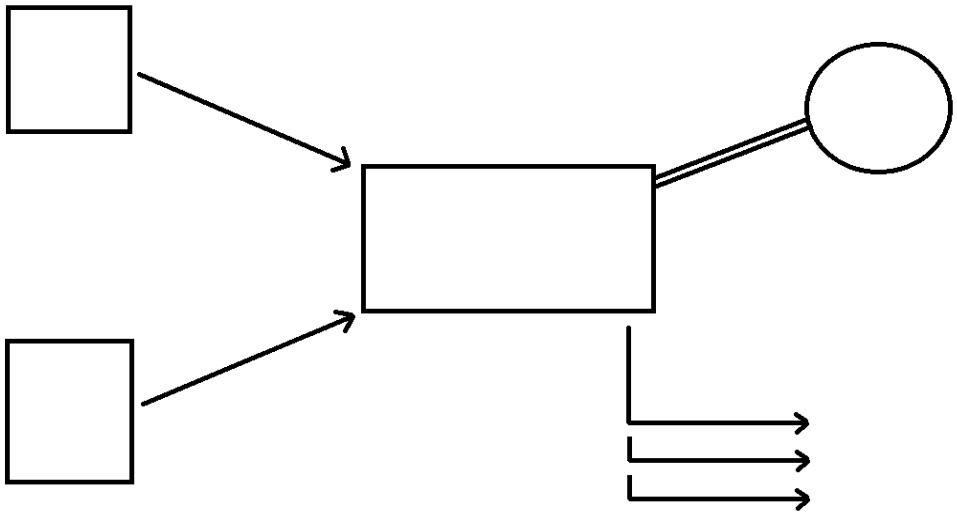

FIGURE 6.1 – Presentation schema

Paragraphe 1
Bla

Paragraphe 2
Bla

Paragraphe 3
Bla

Bibliographie

[1] Documentation Docker https://docs.docker.com/.

[2] Hyper-V Containers https://docs.microsoft.com/en-us/virtualization/windowscontainers.

[3] Linux Containers https://linuxcontainers.org/fr/.

[4] Virtual Box https://www.virtualbox.org/.

[5] Proxmox VE https://www.proxmox.com/en/.

[6] VmWare http://www.vmware.com/fr.html.

[7] QEMU http://www.qemu-project.org/.

[8] Hyper-V https://www.microsoft.com/fr-fr/cloud-platform/server-virtualization.

[9] KVM https://www.linux-kvm.org/.

[10] VM Mark https://www.vmmark.com/products/vmark.html.

[11] VM history par IBM http://www.vm.ibm.com/history/.

[12] An Analysis of Performance Interference Effects in Virtual Environments http://ieeexplore.ieee.org/document/4211036/?arnumber=4211036

[13] The impact of Docker containers on the performance of genomic pipelines https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4586803/

[14] W.Huang, J.Liu, B.Abali, Dhabaleswar K. Panda, "A Case for High Performance Computing with Virtual Machines"

[15] P.Luszczek, E.Meek, S.Moore,D.Terpstra, Evaluation of the HPC Challenge Benchmarks in Virtualized environments, Springer Berlin Heidelberg, coll. « Lecture Notes in Computer Science », 29 août 2011

[16] Documentation de Phoronix test suite. https://gist.github.com/anshula/728a76297e4a4ee7688d.

[17] Site officiel de SaltStack https://saltstack.com/.

[18] Documentation de SaltStack. https://docs.saltstack.com/en/latest/.

[19] https://docs.saltstack.com/en/latest/topics/installation/debian.html.

[20] https://www.phoronix-test-suite.com/documentation/phoromatic.html.