h e p i a

Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·so

Haute Ecole Spécialisée
de Suisse occidentale

MONITORING DE MACHINES VIRTUELLES SOUS LINUX-KVM

MÉMOIRE

Étudiant : Schaub Lionel

Professeur : Gérald Litzistorf

En collaboration avec : HES-SO - Projet Visag

Date: 27 Juin 2011

TABLE DES MATIÈRES

AVANT-PROPOS

RÈGLES TYPOGRAPHIQUES

Ce document a été rédigé avec la police d'écriture Cambria 11pt, en suivant les règles typographiques suivantes:

- Les liens vers des pages web sont en bleu souligné (http://example.com).
- Les LÉGENDES sont en police Cambria 9pt tout en majuscule.
- Les notes de bas de page sont en police Cambria 10pt.
- Le `contenu` des fichiers textes sont en police Courier New 9pt sur fond gris.
- Les `commandes` sont en police Courier New 11pt.
- Les *liens* vers des fichiers ou des dossiers ainsi que des noms propres sont en italique.

Cette icône est présente lorsqu'un point important apparait dans le texte.

TRAVAUX LIÉS

Ce travail est associé au projet Visag de la HES-SO qui utilise la virtualisation pour renforcer la sécurité dans une infrastructure de calcul sur grille. La taille de la grille pouvant être très grande (10-100-1000 hyperviseurs), il a été demandé de développer une solution innovante permettant d'afficher un aperçu de l'état de santé de la grille, de ses hyperviseurs et de ses machines virtuelles.

Ce travail est complémentaire au PA (Projet d'Approfondissement) de M. Sébastien Pasche[1] et à différents travaux de M. Cédric Penas[2].

Vu que M. Pasche a documenté en détail le fonctionnement interne de l'architecture de Linux-KVM et que M. Penas a réalisé des documents sur certains composants (LVM, libvirt, …) utilisés dans mon travail de Bachelor, il y a dans ce document plusieurs liens vers leurs documents.

Certains résultats du travail de M. Pasche ont été utilisés dans ce travail et sont parfois repris dans ce document de manière plus synthétique. Je vous invite donc à suivre les liens vers son document si vous souhaitez plus d'informations.

REMERCIEMENTS

Je tiens à remercier M. Gérald Litzistorf pour m'avoir proposé ce projet de diplôme et m'avoir suivi dans ce travail. Je tiens tout particulièrement à le remercier pour l'aide qu'il m'a apporté dans la rédaction de ce document.

Je remercie également M. Sébastien Pasche et M. Cédric Penas pour leur disponibilité lorsque j'avais des questions en rapport avec leurs travaux respectifs.

Je tiens aussi à remercier toutes les personnes qui ont participé à la relecture de ce document et à tous ceux qui m'ont soutenu de quelque manière que ce soit.

[1] http://www.tdeig.ch/kvm/pasche_R.pdf
[2] http://www.tdeig.ch/kvm/penas.pdf

1. PRÉSENTATION

Le but de ce travail est, dans un premier temps, d'étudier le système de virtualisation Linux-KVM ainsi que les outils permettant de récupérer des statistiques sur l'utilisation (processeur, mémoire RAM, disque, ...) d'un système Linux, puis, dans un second temps, de développer un logiciel de monitoring centralisé des machines virtuelles, fonctionnant sur des hôtes équipés de Linux-KVM.

Ce logiciel doit permettre de donner un aperçu de l'état d'une grille d'hyperviseur via une représentation graphique adaptée.

2. CAHIER DES CHARGES

Développer un logiciel de supervision des machines virtuelles ainsi que des machines hôtes qui composent la grille (Cloud Computing) du projet Visag de la HES-SO.

Le logiciel doit permettre de surveiller l'état de santé de systèmes fonctionnant sur du Linux-KVM.

Définir des scénarios ciblés permettant de valider le fonctionnement du logiciel et de démontrer son utilité.

Produire une documentation des sources du logiciel pour faciliter la reprise du projet par d'autres personnes.

3. NOTIONS IMPORTANTES

Ce chapitre donne une brève description des différents éléments utilisés lors de ce travail ainsi que des liens pointant sur des documents donnant de plus amples informations.

3.1. VIRTUALISATION [3]

La virtualisation consiste à faire fonctionner sur un seul ordinateur physique plusieurs systèmes d'exploitation et de partager ainsi les ressources matérielles.

Un ordinateur physique sur lequel il existe des machines virtuelles est appelé un hyperviseur. Il existe deux types d'hyperviseurs, les hyperviseurs de type 1 et les hyperviseurs de type 2.

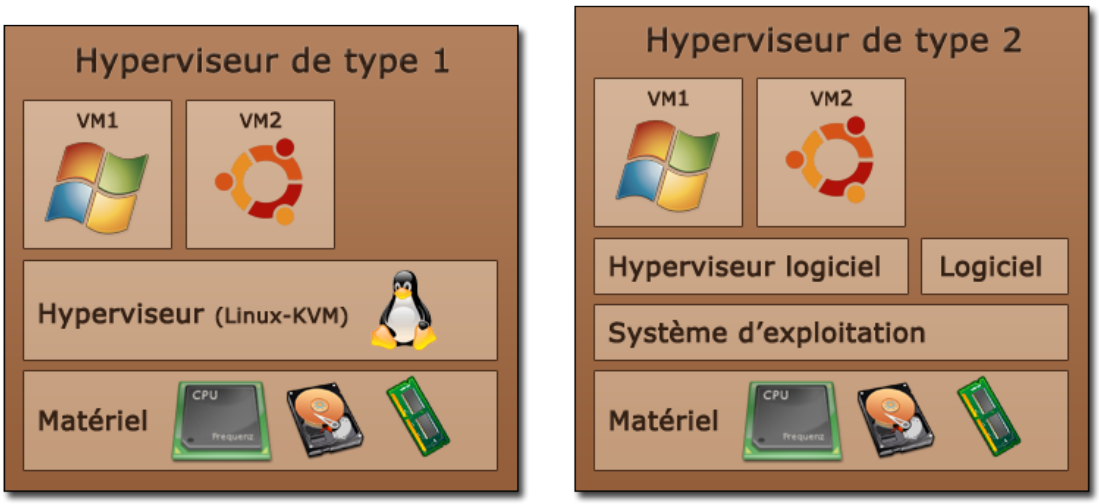


FIGURE 1 - HYPERVISEURS DE TYPE 1 ET 2

Lorsque l'on installe un logiciel (par exemple VMware [4] Workstation) sur un système d'exploitation (par exemple Microsoft Windows) pour virtualiser des machines, c'est un hyperviseur de type 2 (à droite sur la figure ci-dessus) qu'on utilise. Ce genre d'hyperviseur est en général peu performant car il y a un système d'exploitation complet et lourd entre les machines virtuelles et le matériel physique.

Un hyperviseur de type 1(à gauche sur la figure ci-dessus) ne contient qu'un noyau très léger dédié à la gestion des accès des machines virtuelles aux périphériques matériels. Dans ce travail, c'est Linux-KVM qui a été utilisé comme hyperviseur de type 1.

[3] M. Pasche a détaillé cette partie dans le chapitre 2 " Notions d'hyperviseur et de virtualisation " (page 13) de son travail. http://www.tdeig.ch/kvm/pasche_R.pdf
[4] Site officiel de la société VMware http://www.vmware.com/

3.2. LINUX

Linux[5] est un noyau de système d'exploitation qui est libre et open source. Il a été créé en 1991 par Linus Torvalds[6].
Il a ensuite été agrémenté par les logiciels du projet GNU[7].
Des distributions de Linux (Debian, Fedora, Ubuntu, …) sont ensuite apparues et contiennent des logiciels supplémentaires pour former un système d'exploitation.

De nombreux développeurs se sont intéressés à Linux et ont permis de le populariser et de l'améliorer. En effet, de par sa licence libre et son code open source, tout informaticien peut participer à son développement.

Certaines sociétés comme Red Hat[8] vendent leur distribution de Linux et participent activement au développement des nouvelles fonctionnalités de ce système. Red Hat est particulièrement active dans le développement de la solution de virtualisation Linux-KVM[9].

3.3. LVM[10]

LVM (Logical Volume Management) est un système de gestion d'espace disque par volume logiques. Ceci permet une abstraction des périphériques physiques. Il est possible de rassembler plusieurs disques durs en un groupe de volume (Volume Groups) puis de créer des volumes logiques (Logical Volumes) sans se soucier de son emplacement physique. Il est aussi très aisé de redimensionner un LV. Le système LVM permet également d'effectuer des clichés (snapshots) des LVs et ainsi mémoriser l'état du LV afin de pouvoir y revenir lors d'un problème ou d'une mauvaise manipulation.

4. ÉTAPES DE RÉALISATION

Ce travail, qui s'est déroulé sur huit semaines, a été divisé en plusieurs étapes:

1. Prise en main de Fedora, des outils de virtualisation et de mesure des performances.
2. Étude des résultats de ces précédents outils, mise en évidence des valeurs utiles.
3. Conception des scénarios de test.
4. Réalisation des outils permettant de mettre en pratique ces scénarios.
5. Spécification et développement du superviseur.
6. Mise en pratique des scénarios de test.

[5] Site officiel du noyau Linux : http://www.kernel.org/
[6] Linus Torvalds sur Wikipédia : http://fr.wikipedia.org/wiki/Linus_Torvalds
[7] Site officiel du projet GNU : http://www.gnu.org/
[8] Site officiel de Red Hat : http://www.redhat.com/
[9] Site officiel de Linux-KVM : http://www.linux-kvm.org/
[10] Pour plus d'informations sur LVM vous pouvez lire ce document rédigé par M. Cédric Penas : http://www.tdeig.ch/kvm/penas.pdf (chapitre 1).

5. LA VIRTUALISATION SUR LINUX

Il existe plusieurs logiciels libres permettant de virtualiser ou d'émuler un système sous Linux. Nous trouvons parmi les plus connus qemu et Xen. L'intérêt s'est porté sur le premier car il permet, grâce à Linux-KVM, de réaliser un hyperviseur performant à partir d'une distribution Linux.

La figure ci-dessous représente l'architecture de l'hyperviseur de type 1 à base de Linux-KVM qui a été utilisé dans ce travail. Les différents éléments de cette architecture sont détaillés dans les sections suivantes.

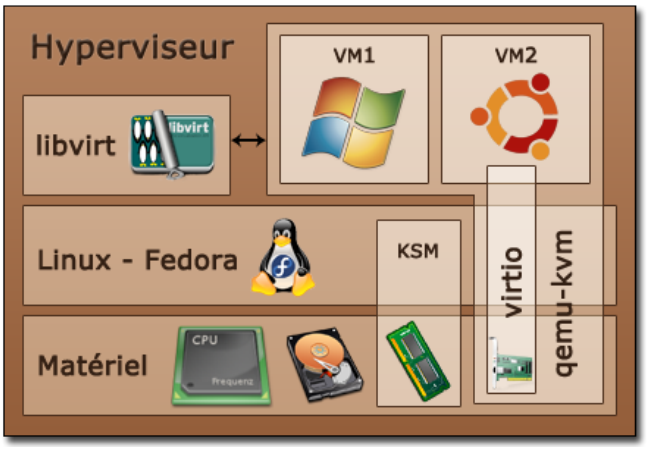


FIGURE 2 - COMPOSANTS FORMANT UN HYPERVISEUR LINUX-KVM

5.1. CHOIX D'UNE DISTRIBUTION

La distribution Fedora[11] a été utilisée dans ce travail car elle contient les dernières nouveautés autour de la virtualisation avec Linux-KVM. De plus, les développeurs de cette technologie et de ses outils utilisent Fedora comme plateforme cible.

Cette distribution est la version communautaire de Red Hat Entreprise Linux dont Red Hat se sert comme processus de développement ouvert. Comme Red Hat est très actif dans le développement de Linux-KVM, cette distribution est donc totalement appropriée.

5.2. QEMU-KVM

Qemu[12] est en réalité un émulateur qui permet d'émuler différents types de matériel tel que l'architecture x86, PowerPC et SPARC. Comme c'est un émulateur, les machines virtuelles seront, par définition, sensiblement plus lentes. De telles performances ne permettraient pas d'exploiter ce logiciel en production, néanmoins grâce à qemu-kvm ces limitations sont outrepassées.

[11] Site web officiel du projet Fedora: http://fedoraproject.org/
[12] Site web officiel de l'émulateur qemu: http://www.qemu.org/
Article de qualité sur qemu(-kvm) sur le wiki d'archlinux : http://wiki.archlinux.fr/Qemu

En effet, qemu-kvm est un logiciel qui réutilise une grande partie des fonctions de qemu sans toutefois être un émulateur. Il profite, grâce au module noyau KVM, d'une accélération matérielle fournie par les processeurs modernes via le jeu d'instructions AMD-V ou Intel VT.

Avec qemu-kvm, chaque machine virtuelle correspond à un processus sur la machine hôte (hyperviseur).
On trouve dans ce processus plusieurs éléments dont, la machine virtuelle et un serveur VNC qui permet de la contrôler à distance. Il y a également plusieurs threads, un par cœur de chaque processeur virtuel, ainsi que d'autres threads qui sont créés et détruits à la volée pour effectuer des opérations d'E/S.

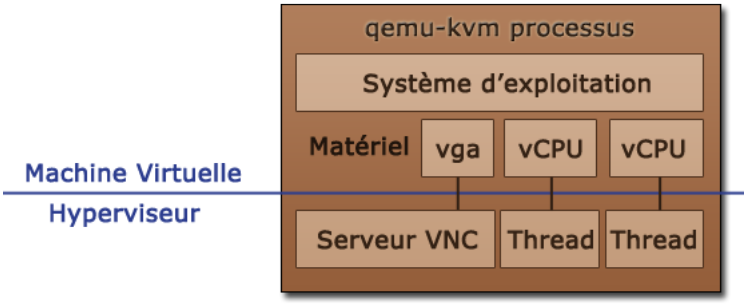


FIGURE 3 - PROCESSUS QEMU-KVM

Le contrôle de qemu-kvm n'est pas aisé, il faut créer certains composants manuellement, puis démarrer la machine virtuelle en exécutant la commande qemu-kvm avec en paramètre toute la configuration de la machine. Ceci donne une commande qui ressemble à cela : `/usr/bin/qemu-kvm -S -M pc-0.13 -enable-kvm -m 1024 -smp 2,sockets=2,cores=1,threads=1 -name 2-Fedora-14-64bits -uuid 6cdbd6d2-266d-c568-bc78-7aaab689c0e5 -nodefconfig -nodefaults -chardev socket,id=charmonitor,path=/var/lib/libvirt/qemu/2-Fedora-14-64bits.monitor,server,nowait -mon chardev=charmonitor,id=monitor,mode=control -rtc base=utc -boot order=c,menu=off -drive file=/dev/vg_fedora14/2-Fedora-14-64bits,if=none,id=drive-virtio-disk0,boot=on,format=raw -device virtio-blk-pci,bus=pci.0,addr=0x5,drive=drive-virtio-disk0,id=virtio-disk0 -drive if=none,media=cdrom,id=drive-ide0-1-0,readonly=on,format=raw -device ide-drive,bus=ide.1,unit=0,drive=drive-ide0-1-0,id=ide0-1-0 -netdev tap,fd=21,id=hostnet0 -device rtl8139,netdev=hostnet0,id=net0,mac=52:54:00:e2:8c:ce,bus=pci.0,addr=0x3 -chardev pty,id=charserial0 -device isa-serial,chardev=charserial0,id=serial0 -usb -device usb-tablet,id=input0 -vnc 127.0.0.1:2 -vga cirrus -device AC97,id=sound0,bus=pci.0,addr=0x4 -device virtio-balloon-pci,id=balloon0,bus=pci.0,addr=0x6.`

5.3. LIBVIRT[13]

C'est pour simplifier cette gestion des VMs que le projet libvirt est né. Libvirt est une API libre qui permet à des programmes d'utiliser ses fonctions génériques de contrôle de solution de virtualisation tels que qemu, Xen, VMware ESXi, etc. Ces solutions peuvent se trouver sur la machine locale ou sur une machine distante.

Virsh est un Shell qui utilise libvirt pour exécuter des commandes sur les machines virtuelles. La définition des paramètres des différents éléments (machines virtuelles, réseaux, …) est effectuée par le biais de fichiers XML.

Libvirt propose également un outil graphique (virt-manager) qui permet à l'utilisateur novice de créer et de contrôler des machines virtuelles en toute simplicité.

[13] Site officiel de libvirt: http://www.libvirt.org/

Dans le cadre de ce travail j'ai utilisé libvirt avec le langage Python[14] pour récupérer des informations sur des machines virtuelles tournant sur qemu-kvm.

5.4. RÉSEAUX VIRTUELS[15]

La virtualisation sur Linux nous permet de réaliser un parc informatique complet sur une ou plusieurs machines physiques. Il est possible de créer des réseaux virtuels pour les VMs et de créer des ponts (bridges) entre ces réseaux. Il est également possible de relier un réseau ou des machines virtuelles directement sur une interface réseau physique.

5.5. PARAVIRTUALISATION

La paravirtualisation est une notion qui n'est pas directement utilisée dans ce projet, elle est néanmoins importante pour l'optimisation des performances des machines virtuelles.

Il existe des pilotes dédiés pour les machines virtuelles qui prennent en compte l'aspect virtuel des machines et qui permettent d'optimiser considérablement les performances. Cette suite de pilotes se nomme Virtio[16] et contient un pilote pour les accès disque, un autre pour le réseau et un dernier pour le système de ballooning.

Ces pilotes prennent en compte, par exemple, qu'une communication réseau entre deux machines virtuelles sur le même hyperviseur correspond à une communication inter-processus[17].

Le driver de ballooning permet à l'hyperviseur de demander de la mémoire RAM à une VM et à une VM de demander de la RAM à l'hyperviseur. Ce mécanisme est très puissant et permet par exemple de diminuer à chaud la quantité de mémoire d'une machine virtuelle.

Ces pilotes doivent néanmoins être installés sur la machine virtuelle. On parle alors de paravirtualisation. Les pilotes Virtio sont inclus dans les noyaux Linux récents et il en existe également une version pour les systèmes Windows.

5.6. PRIORITÉS D'ACCÈS AUX RESSOURCES

Comme chaque machine virtuelle correspond à un processus sur l'hyperviseur, ceci permet d'utiliser les mécanismes d'ordonnancement[18] de Linux pour donner des priorités aux machines virtuelles. On peut actuellement donner des priorités pour le CPU et les disques durs grâce aux commandes `nice`[19] et `ionice`[20].

[14] Site officiel de Python: http://www.python.org/
[15] Annexe A.4 : Création d'un bridge réseau
[16] Virtio sur le site web de Linux-KVM : http://www.linux-kvm.org/page/Virtio
Virtio sur le site web de libvirt : http://wiki.libvirt.org/page/Virtio
[17] Article *Communication inter-processus* sur Wikipédia :
http://fr.wikipedia.org/wiki/Communication_inter-processus
[18] Document détaillant l'ordonnancement CPU sous Linux :
http://jve.linuxwall.info/ressources/taf/linux_ps-et-scheduler.pdf
[19] Page de manuel de nice : http://www.kernel.org/doc/man-pages/online/pages/man2/nice.2.html
[20] Page de manuel d'ionice : http://linux.die.net/man/1/ionice

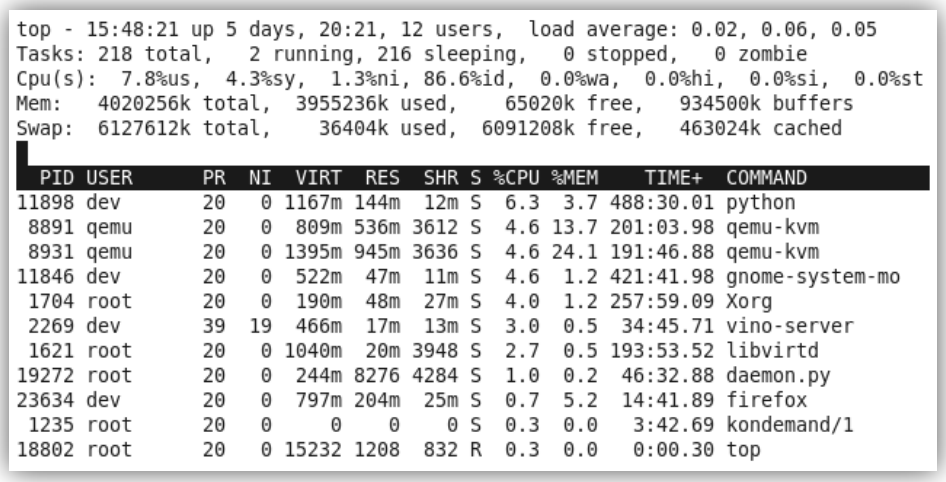
5.7. OVERCOMMITMENT

L'overcommitment est une notion qui n'est pas directement utilisée dans ce projet, elle est néanmoins utile pour comprendre pourquoi il est possible que les machines virtuelles utilisent une plus grande quantité de mémoire que celle utilisée par l'hyperviseur.

Il existe un système nommé KSM qui parcourt la mémoire RAM de l'hyperviseur à la recherche de pages mémoires identiques pour ensuite les fusionner et ainsi libérer de la mémoire. Ce mécanisme, couplé au système de ballooning, permet d'utiliser plus de mémoire que celle disponible sur l'hyperviseur.

6. ÉTUDE DES DIFFÉRENTS OUTILS DE MESURE D'UN SYSTÈME LINUX

Linux dispose d'une multitude d'outils[21] (top, iostat, free, ...) qui vont interroger le noyau pour obtenir des statistiques système (charge CPU, utilisation mémoire, accès disque, statistiques NFS, ...). Certains de ces outils sont intégrés dans la plupart des distributions, d'autres sont disponibles en tant que paquets.



```
top - 15:48:21 up 5 days, 20:21, 12 users,  load average: 0.02, 0.06, 0.05
Tasks: 218 total,   2 running, 216 sleeping,   0 stopped,   0 zombie
Cpu(s):  7.8%us,  4.3%sy,  1.3%ni, 86.6%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4020256k total,  3955236k used,    65020k free,   934500k buffers
Swap:  6127612k total,    36404k used,  6091208k free,   463024k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
11898 dev       20   0 1167m 144m  12m S  6.3  3.7 488:30.01 python
 8891 qemu      20   0  809m 536m 3612 S  4.6 13.7 201:03.98 qemu-kvm
 8931 qemu      20   0 1395m 945m 3636 S  4.6 24.1 191:46.88 qemu-kvm
11846 dev       20   0  522m  47m  11m S  4.6  1.2 421:41.98 gnome-system-mo
 1704 root      20   0  190m  48m  27m S  4.0  1.2 257:59.09 Xorg
 2269 dev       39  19  466m  17m  13m S  3.0  0.5  34:45.71 vino-server
 1621 root      20   0 1040m  20m 3948 S  2.7  0.5 193:53.52 libvirtd
19272 root      20   0  244m 8276 4284 S  1.0  0.2  46:32.88 daemon.py
23634 dev       20   0  797m 204m  25m S  0.7  5.2  14:41.89 firefox
 1235 root      20   0     0    0    0 S  0.3  0.0   3:42.69 kondemand/1
18802 root      20   0 15232 1208  832 R  0.3  0.0   0:00.30 top
```

FIGURE 4 - CAPTURE D'ÉCRAN DE L'OUTIL TOP

Pour surveiller l'état des machines virtuelles ainsi que leurs systèmes hôtes, il faut choisir parmi tous les outils disponibles ceux qui livrent les informations les plus pertinentes.

Parmi une vingtaine d'outils étudiés, 4 ont été retenus:

1. `free` qui nous informe sur l'utilisation mémoire du système.
2. `iostat` qui permet de récupérer le taux d'utilisation des disques.
3. `top` qui nous renvoie des informations sur la charge CPU.
4. `virt-top` qui donne des informations sur les machines virtuelles du système tels que la charge CPU, l'utilisation de la mémoire, les accès réseau et les accès disque.

L'outil `iostat` effectue des mesures pendant un certain laps de temps mais affiche un premier résultat instantanément qui est donc erroné. Il faut lui spécifier d'afficher au minimum deux résultats et il ne faut pas prendre en compte le premier.

Tous ces outils ont un point commun. En effet, ils vont tous interroger le noyau grâce au pseudo système de fichier *proc*[22]. Ils vont ensuite effectuer certains calculs et reformater les données pour, au final, les afficher.

Dans la réalisation de ce projet, ces commandes (virt-top, iostat, free, ...) n'ont pas été utilisées car les valeurs nécessaires sont calculées à partir des informations récupérées grâce à *proc*. Ceci permet d'éviter de lancer un processus tel qu'`iostat` qui serait bloquant pendant un

[21] Les 20 outils que les administrateurs systèmes devraient connaître : http://www.cyberciti.biz/tips/top-linux-monitoring-tools.html
[22] Annexe A.2 : Le pseudo système de fichier *proc*

temps prédéfini et par conséquent, ça évite l'utilisation de threads lors de la récupération des informations. Ça permet également d'éviter le calcul de valeurs qui ne sont pas nécessaires.

7. MONITORING D'UNE MACHINE LINUX

Lorsque l'on veut développer un système de surveillance des machines virtuelles et de leurs hôtes, il faut réfléchir aux éléments que l'on veut mesurer ainsi qu'aux indicateurs appropriés pour obtenir une vue correcte de l'état de santé des machines.

J'ai choisi, en accord avec mon professeur, de mesurer la charge des principaux éléments matériels, soit le processeur, la mémoire et le disque dur.

7.1. PROCESSEUR

En ce qui concerne la charge CPU, les données fournies par *proc* sont peu révélatrices. En effet, un certain nombre d'informations fournies par *proc* sont données en temps CPU, c'est-à-dire le temps passé à exécuter du code dans le processeur depuis le démarrage du système. À première vue ces valeurs ne sont pas intéressantes, pourtant si l'on compare deux temps CPU acquis à des moments différents, il est possible de calculer le taux d'utilisation du CPU.

Prenons, par exemple, le cas où l'on veut calculer l'utilisation du processeur par un processus. Si l'on prend deux mesures espacées d'une seconde et qu'en les soustrayant on obtient 0.5[sec], cela signifie que le processus utilise 50% de la capacité du processeur. Voici la formule pour effectuer ce calcul : $utilCPU[\%] = \frac{tempsCPU_B - tempsCPU_A}{temps_B - temp_A} * 100$
Pour obtenir les valeurs $tempsCPU_B$ et $tempsCPU_A$ il faut lire le fichier `/proc/PID/stat`.

7.2. MÉMOIRE RAM

Le noyau Linux nous donne beaucoup d'informations sur l'utilisation de la mémoire grâce à `/proc/meminfo`. On obtient une quarantaine de valeurs, néanmoins seulement deux valeurs semblent intéressantes, la mémoire totale (`MemTotal`) et la mémoire libre (`MemFree`). On peut grâce à ces deux valeurs calculer l'utilisation mémoire grâce à la formule suivante : $utilisation[\%] = \frac{(MemTotal - MemFree)}{MemTotal} * 100$

En réalité, cette formule n'est pas correcte car le noyau prend en compte les buffers et le cache qui résident en RAM mais qui ne sont pas nécessaires et qui peuvent être supprimés à tout moment par le noyau. Il n'est donc pas pertinent de les prendre en compte. Pour cela il faut également lire les valeurs `Buffers` et `Cached`.
La formule devient donc: $utilisation[\%] = \frac{(MemTotal - MemFree - Buffers - Cached)}{MemTotal} * 100$

7.3. DISQUE DUR

Plusieurs valeurs ont été étudiées pour la mesure de la charge du disque dur telles que le nombre de blocs lu et écrits par seconde, la quantité de données lues et écrites par seconde, le temps de latence, la longueur de la file d'attente, etc.
Toutes ces valeurs sont dépendantes des performances[23] du disque dur. Elles ne permettent donc pas de créer un logiciel fonctionnant sur n'importe quel disque.
La longueur de la file d'attente peut, à première vue, paraître une bonne valeur, néanmoins elle ne l'est pas car le noyau Linux place souvent des demandes d'E/S dans la file d'attente pour les exécuter à un moment plus approprié.

C'est la valeur du taux d'utilisation qui a été retenue pour mesurer la charge d'un disque dur. En effet cette valeur correspond au pourcentage de temps que le processeur passe dans des opérations d'E/S disque. Ceci la rend indépendante des caractéristiques techniques du disque dur ainsi que de la mécanique interne du noyau. Pour obtenir cette valeur il faut mesurer le temps pendant lequel il y a au moins un processus qui est dans l'état "Suspendu" (voir la figure ci-dessous), puis la diviser par le temps pendant lequel on a effectué cette mesure.

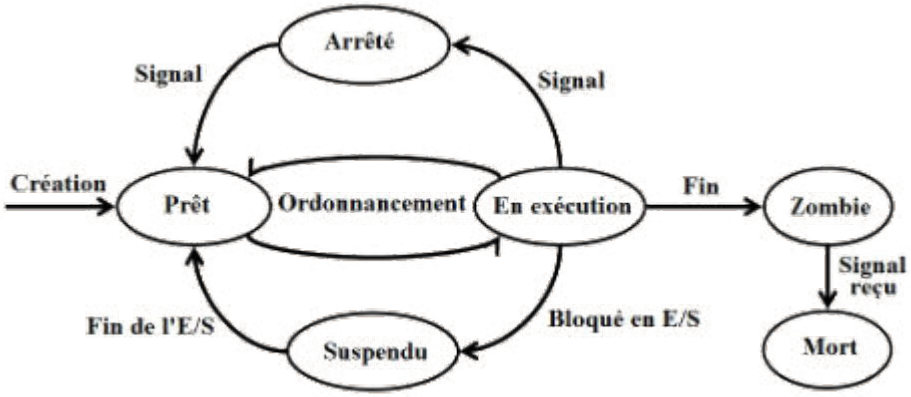


FIGURE 5 - ÉTATS D'UN PROCESSUS SOUS LINUX

Pour mesurer ce pourcentage, il faut définir un intervalle de mesure qui peut faire varier sensiblement les résultats. En effet, si l'on prend un intervalle très court (par exemple 10ms) la mesure sera soit de 0% soit de 100% car pendant une telle durée, le processeur est soit en train d'effectuer un accès disque soit en train de faire autre chose. Après avoir effectué des tests, il s'avère que trois secondes soit une bonne valeur.

[23] Le chapitre 8. Benchmarking de disque dur donne des informations sur la méthode de quantification de ces performances.

8. BENCHMARKING DE DISQUE DUR

La réflexion sur les indicateurs de la charge du disque dur a soulevé deux questions importantes.

8.1. QUELLES DONNÉES PERMETTENT DE QUANTIFIER LA PERFORMANCE D'UN DISQUE DUR ?

Les sociétés qui se sont spécialisées dans les benchmarks de composants matériels, utilisent pour leurs tests, trois des multiples valeurs fournies par les principaux logiciels[24] de benchmark tel que Iometer, développé à l'origine par Intel et FIO (Flexible I/O Tester), développé par le créateur de la plupart des ordonnanceurs d'E/S des systèmes Linux dont celui par défaut.

Ces valeurs sont la latence, le débit utile ainsi que le nombre d'opérations d'E/S par seconde. Ces valeurs n'ont pas été choisies par hasard, elles sont dépendantes des caractéristiques matérielles et décrivent les performances pour un type d'utilisation donné.

Par exemple, si l'on souhaite effectuer beaucoup de petites écritures, il faut se préoccuper du nombre d'E/S par seconde. Si l'on souhaite effectuer des copies de gros fichiers, il faut se préoccuper principalement du débit utile.

Il est également important de fournir ces valeurs en fonction de la taille du bloc que l'on lit/écrit. Il faut aussi préciser si l'on fait une lecture/écriture séquentielle ou aléatoire (s'applique uniquement aux disques mécaniques).

8.2. QUEL EST LA MÉTHODOLOGIE DE MESURE ?

La méthodologie de mesure des différents outils de benchmark de disque dur est en réalité relativement simple.

Ces outils chargent le disque dur en lui envoyant des commandes de bas niveau qui vont effectuer des lectures et des écritures avec un bloc d'une certaine taille.
Le logiciel va effectuer ces opérations autant de fois que possible et compter le nombre d'opérations qu'il a effectué. À chaque seconde, il va pouvoir déterminer le nombre d'opérations effectuées (IOPS) ainsi que le débit (grâce à la taille du bloc). Le calcul de la latence se fait grâce à un thread qui mesure le temps pendant que le programme est bloqué sur une opération d'E/S.

[24] Annexe A.5 : Installation de Iometer et de fio sur Linux

9. SCÉNARIOS

Afin de valider le bon fonctionnement et l'utilité du système de monitoring qui a été développé dans le cadre de ce travail, des scénarios ciblés ont été mis au point.

Certains scénarios permettent de simuler des cas (problèmes) qui pourraient survenir lors d'une utilisation en production, d'autres permettent de vérifier que les valeurs sont correctement mesurées, acheminées et affichées dans le logiciel de supervision.

Les éléments qui sont visés par ces scénarios sont l'utilisation du processeur, l'utilisation du disque dur ainsi que l'utilisation de la mémoire RAM. Chaque test ne s'occupe que d'un seul de ces éléments pour ne pas subir l'influence qu'un test sur un composant peut avoir sur un autre test.
En revanche, certains scénarios seront effectués sur plusieurs machines virtuelles pour vérifier l'augmentation de la charge de l'hyperviseur en fonction de l'augmentation de la charge des VMs.

Les scénarios suivants ont été conçus:

- CPU
  1. Faire varier la charge CPU à des valeurs connues pour vérifier les valeurs obtenues grâce à libvirt (pour les VMs) et grâce à */proc* (pour l'hôte).
  2. Simuler la montée en charge simultanée de plusieurs serveurs (VMs).
- RAM
  1. Faire varier la charge RAM à des valeurs connues pour vérifier les valeurs obtenues grâce à libvirt (pour les VMs) et grâce à */proc* (pour l'hôte).
  2. Charger à presque 100% toutes les VMs.
- Disque Dur
  1. Tenter de charger le disque à des valeurs connues pour vérifier les valeurs obtenue grâce à */proc*.
  2. Charger le disque dur d'une VM et observer si on remarque un changement de pourcentage d'utilisation du disque sur une autre VM.
  3. Charger le disque d'une VM à cinquante pourcent puis charger le disque d'une autre VM au maximum et observer si le pourcentage d'utilisation du disque sur la première VM augmente.

Pour mettre en œuvre ces scénarios, il faut des outils permettant de générer de la charge au niveau du processeur de la mémoire RAM et du disque dur. Comme ces outils n'existent pas ou ne correspondent pas aux exigences, ils ont été développés.

Pour chaque composant pour lequel on veut générer une charge, correspond un outil. Ces logiciels prennent en argument la valeur de charge à générer et un temps d'exécution. Ceci permet de créer des fichiers de script qui exécuteront plusieurs fois un des logiciels avec différentes valeurs de charge.

```
[root@Fedora14 dev]# ./memoryLoadGenerator2
Entrez la charge mémoire à maintenir [Mo]: 3000
Entrez le temps d'exécution du script (-1 pour infini) [s]: 30
SysMem: 2556544000
Taille: 3000000000
Ajout: 443456000
Size: 530 = 503 + 26   Diff: 3000 - 2973
Size: 529 = 530 + 0   Diff: 3000 - 3000
Size: 530 = 529 + 0   Diff: 3000 - 2999
Size: 530 = 530 + 0   Diff: 3000 - 2999
Size: 529 = 530 + 0   Diff: 3000 - 3000
Size: 529 = 529 + 0   Diff: 3000 - 3000
Size: 526 = 529 + -3   Diff: 3000 - 3003
Size: 526 = 526 + 0   Diff: 3000 - 2999
Size: 526 = 526 + 0   Diff: 3000 - 3000
```

FIGURE 6 - EXÉCUTION DU LOGICIEL MEMORYLOADGENERATOR2

En passant '-h' ou '--help' en argument de ces outils, une aide s'affiche.

Pour générer une charge CPU en escalier qui passe par 30%, 60% et 90% et pour que chaque étape dure 10 secondes, il faut écrire ceci dans un fichier de script :

```
./cpuLoadGenerator.py 30 10
./cpuLoadGenerator.py 60 10
./cpuLoadGenerator.py 90 10
```

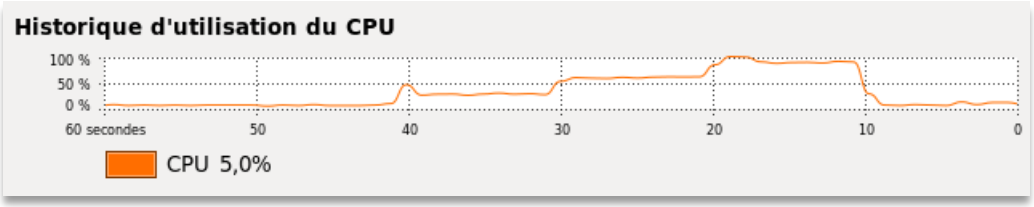


FIGURE 7 - RÉSULTAT DE L'EXÉCUTION DU SCRIPT CI-DESSUS

Il a également été envisagé d'utiliser un système de benchmark de disque dur, tel qu'Iometer ou fio, pour générer une charge maximale sur le disque dur. Ce scénario a été abandonné car il ne permet pas de contrôler la charge générée et ces outils ne prennent pas forcément en compte les données présentes sur le disque et vont détruire le système présent sur ce dernier disque.

10. ARCHITECTURE DU SUPERVISEUR

On peut voir sur la figure ci-dessous l'apparence du produit final:

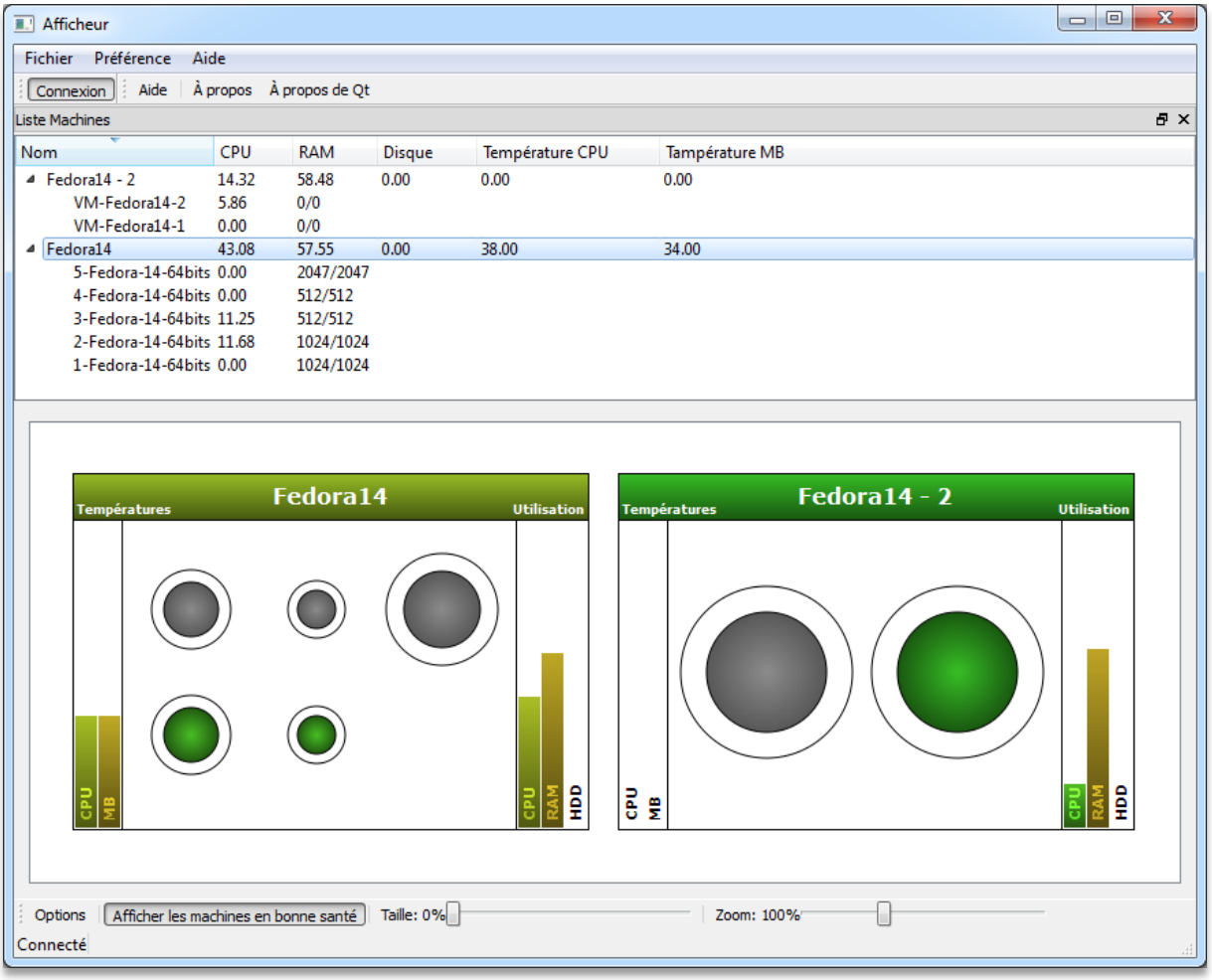


FIGURE 8 - CAPTURE D'ÉCRAN DE L'AFFICHEUR

Néanmoins, avant d'arriver à ce résultat, plusieurs étapes ont dû être effectuées.

À commencer par la réflexion sur l'architecture du superviseur qui n'est pas triviale car il faut choisir parmi la multitude de possibilités que l'informatique nous offre, celles qui seront utilisées. Il faut également que les choix que l'on fait soient compatibles entre eux.

Il faut étudier la disponibilité des API dans divers langages de programmation, les possibilités offertes par le langage, le type de communication entre les différentes machines, etc.

Il faut également déterminer si c'est le client qui va demander les données aux hyperviseurs (en brun sur la figure ci-dessous) ou si c'est les hyperviseurs qui vont envoyer les données en continu au client.

Lors de la réflexion sur l'architecture à adopter, un accent particulier a été mis sur ces points:

1. La modularité du système ainsi que la possibilité de le faire évoluer en ajoutant des composants.

2. La possibilité d'ajouter le plus simplement possible une sécurité dans les transactions réseaux.
3. Réaliser un système simple et puissant.

En prenant en compte tous ces paramètres, il en est ressorti qu'il fallait séparer le superviseur en trois parties:

1. Le **daemon** qui va récupérer les données sur l'hyperviseur et ses machines virtuelles pour les envoyer au(x) collecteur(s).
2. Le **collecteur** qui va recevoir et stocker les données en provenance des hyperviseurs, puis les envoyer aux afficheurs connectés.
3. L'**afficheur** qui va recevoir les données du collecteur puis les afficher sous forme de graphique.

J'ai choisi de développer le daemon en Python car c'est un langage qui est intégré sur la plupart des systèmes Linux. Il est également pris en charge par l'API libvirt. Il contient une grande librairie standard ce qui simplifie le développement.

J'ai choisi de réaliser le collecteur et l'afficheur en C++ avec le Framework Qt pour sa capacité à générer des exécutables sur de multiples plateformes telles que Linux, Windows et Mac OS X. La librairie Qt est très complète et son système de signaux et de slots est très puissant.

La figure ci-dessous représente l'utilisation du superviseur avec deux hyperviseurs, et un utilisateur (1) qui surveille les hyperviseurs grâce à une machine dans le réseau local. Il y a également un deuxième utilisateur (2) qui est connecté au collecteur depuis un ordinateur distant. Un collecteur de secours est aussi présent.

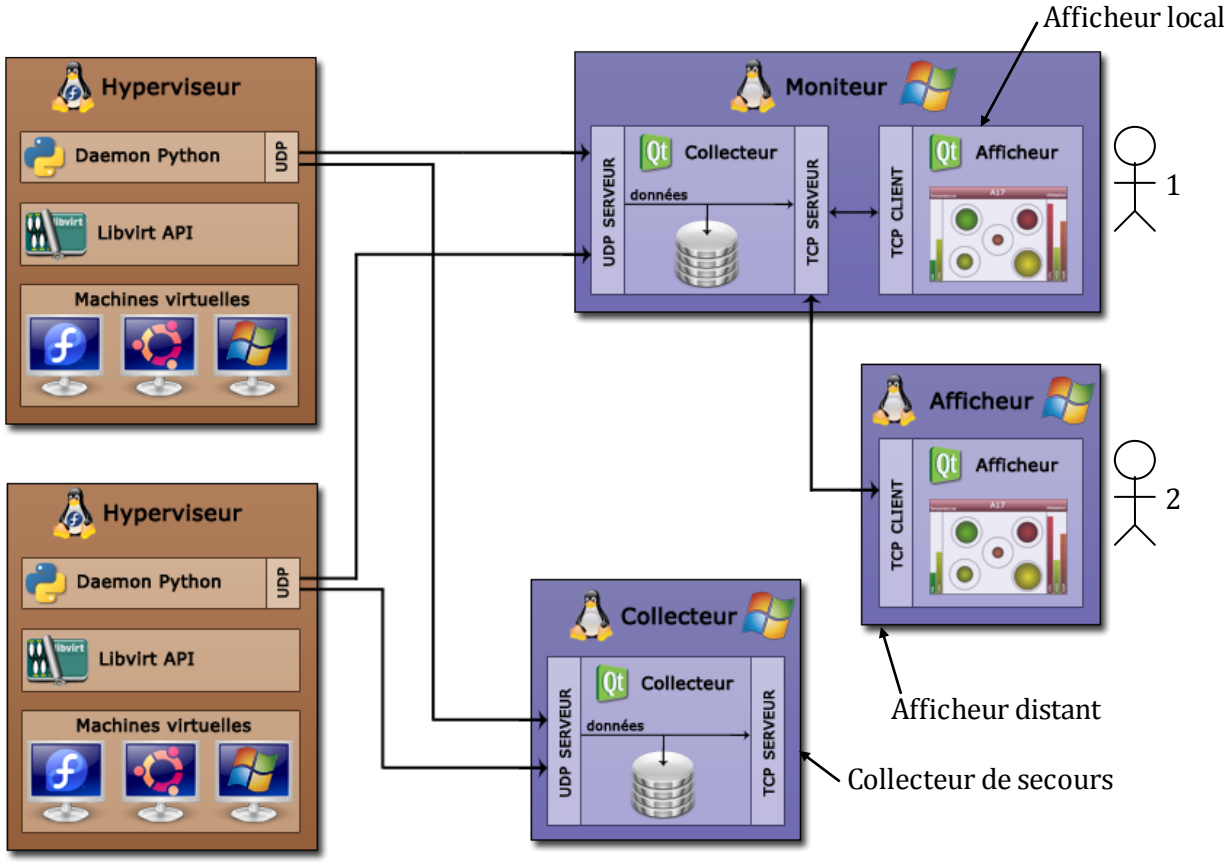


FIGURE 9 - ARCHITECTURE DU SUPERVISEUR

10.1. COMMUNICATIONS

Les communications entre les daemons et les collecteurs sont effectuées grâce à des sockets UDP tandis que celles entre les collecteurs et les afficheurs sont effectuées grâce à des sockets TCP. L'utilisation de sockets de bas niveau permet d'utiliser n'importe quel langage aux deux extrémités de la liaison.

Les données envoyées par le daemon sont transmises au format JSON (ce format est détaillé dans la section suivante). Les données JSON ne sont pas modifiées par le collecteur lorsqu'il les relaye à l'afficheur.

Le daemon envoi des données à intervalle régulier et le reste du système est synchronisé par les données.

La figure ci-dessous montre un exemple d'échange. On y voit que dès que l'afficheur se connecte, le collecteur lui transmet les données en provenance du daemon.

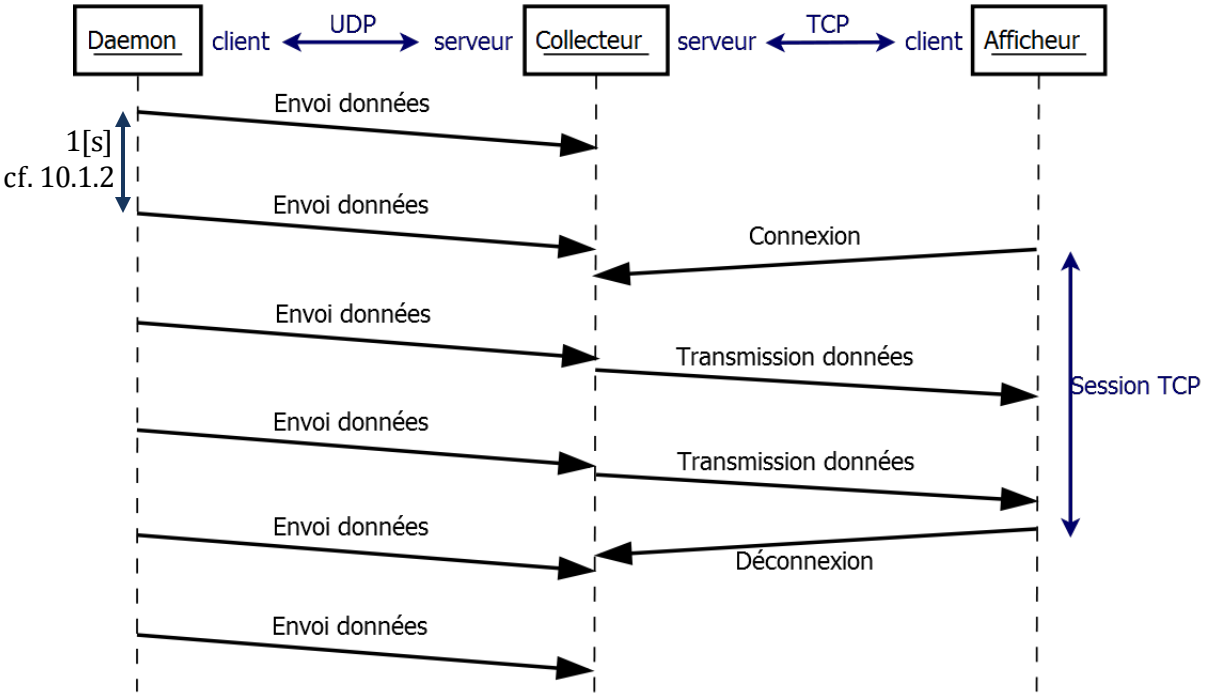


FIGURE 10 - COMMUNICATIONS ENTRE LES TROIS SOUS-PROGRAMMES

10.1.1. LE FORMAT JSON

Le format JSON[25] (JavaScript Object Notation) est un format léger d'échange de données. Il est facile à comprendre et à visualiser par l'Homme grâce à son format texte. Il est également facile à parser.

Dans le cadre de son utilisation pour le superviseur il contient des ensembles de paires nom/valeur et un tableau contenant également des ensembles de paires nom/valeur pour chaque machine virtuelle.

[25] RFC du format JSON: http://tools.ietf.org/html/rfc4627
Site officiel : http://www.json.org/

L'ordre des éléments n'a pas d'importance, y compris pour le tableau.

La chaîne JSON ci-dessous a été reformatée pour être plus lisible. Lors de son envoi via le réseau, elle ne contient pas de retour à la ligne, de commentaires et de tabulations.

```
{
     "cputemp": 37.0, // commande sensors
     "mbtemp": 36.0,  // idem
     "ram": 17.27959612522188, // /proc/meminfo
     "vms": [ // libvirt
            {
                   "maxram": 1048576,
                   "name": "1-Fedora-14-64bits",
                   "ram": 1048576,
                   "state": 5,
                   "cpu": 0.0,
                   "uuid": "87c63557-2a88-d1ee-b38b-9d7211f0256d"
            }, {
                   "maxram": 1048576,
                   "name": "2-Fedora-14-64bits",
                   "ram": 1048576,
                   "state": 5,
                   "cpu": 0.0,
                   "uuid": "6cdbd6d2-266d-c568-bc78-7aaab689c0e5"
            }
     ],
     "time": 1308674056.734489,
     "disk": 0.0,  // /proc/diskstats
     "cpu": 5.350496117327408, // /proc/stat
     "name": "Fedora14" // /etc/sysconfig/network
}
```

EXEMPLE D'UNE CHAÎNE JSON ENVOYÉ PAR UN HYPERVISEUR AVEC DEUX MACHINES VIRTUELLES.

10.1.2. LIAISON DAEMON-COLLECTEUR

L'utilisation d'une connexion UDP est très adaptée pour la liaison daemon-collecteur car c'est un protocole orienté message, sans connexion, qui n'a besoin que d'un paquet pour envoyer des données, tandis que le protocole TCP a besoin d'au moins six paquets (plus que deux paquets après que la connexion a été établie) pour effectuer le même travail (voir la figure ci-dessous).

Le protocole UDP ne permet pas de savoir si le paquet a été reçu et il peut y avoir des pertes de paquets, mais ceci n'est pas important pour cette liaison car le daemon envoie les données en continu. Par conséquent, si un paquet est perdu, un autre va suivre avec les données les plus récentes.

La trame UDP est constituée d'un numéro de séquence sur 32 bits, suivi d'une chaîne de caractère au format JSON. Le numéro de séquence est incrémenté à chaque envoi et permet au

collecteur de détecter des pertes de paquets. Ce numéro est codé selon le mode big-endian et respecte le standard *network byte order* [26]du protocole IP.

FIGURE 11 - FORMAT DES DONNÉES UDP

Le délai entre deux envois de données est paramétrable dans le daemon et est configuré, par défaut, à une seconde. Ce délai a une influence directe sur la charge CPU [27] et la charge disque [28] car ces valeurs sont mesurées sur un intervalle de temps.

Il faut néanmoins garder à l'esprit qu'il est possible qu'un paquet A envoyé avant un paquet B arrive après le paquet B. Ce phénomène peut se produire si les deux paquets empruntent un chemin différent et que le chemin du paquet A est plus lent.

10.1.3. LIAISON COLLECTEUR-AFFICHEUR

La liaison entre le collecteur et l'afficheur utilise une connexion TCP pour permettre à l'afficheur de fournir une meilleure expérience utilisateur en indiquant, par exemple, si la connexion est établie ou si elle a échouée. L'utilisation du protocole TCP permet également de profiter des mécanismes de connexion pour que le collecteur sache quand il doit envoyer les données (lorsque la connexion est établie) et quand il doit s'arrêter (lorsque la connexion est rompue).

Il a été prévu d'implémenter, dans une version future, un mécanisme permettant d'interroger le collecteur pour récupérer des anciennes données et/ou des statistiques. Dans ce cas, il est préférable d'utiliser un protocole avec gestion des connexions.

La trame TCP est constituée d'un nombre codé sur 32 bits (selon le mode big-endian) représentant la taille du message, suivi d'un champ réservé de 8 bits, suivi d'une chaîne de caractères au format JSON. Le champ réservé n'est pas utilisé, il a été prévu pour contenir un numéro identifiant le type de message dans le cas où plusieurs types de messages transiteraient entre le collecteur et l'afficheur.

FIGURE 12 - FORMAT DES DONNÉES TCP

10.1.4. QUELQUES CHIFFRES

L'utilisation de l'UDP permet de légèrement diminuer l'utilisation de la bande passante.

La taille du message dépend du nombre de machines virtuelles sur l'hyperviseur car pour chaque VM il faut envoyer des informations.

---

[26] Plus d'informations sur Wikipédia : http://fr.wikipedia.org/wiki/Endianness#Dans_les_communications
[27] Pour plus d'informations voir 7.1. Processeur
[28] Pour plus d'informations voir 7.3. Disque dur

Il est possible d'estimer l'utilisation de la bande passante grâce à cette formule:
(Le délai correspond au temps entre deux échantillons)

$$utilisation\ bande\ passante\ \left[\frac{Ko}{s}\right] \cong 200 * \frac{(nbVM + 1)}{\text{délai}[s]}$$

La capture Wireshark ci-dessous, montre les échanges entre les trois éléments composant le superviseur. On y voit le daemon (192.168.1.15) qui envoie des données en UDP au collecteur (192.168.1.10) et le collecteur qui renvoie ces données en TCP à un afficheur (192.168.1.200) connecté.

Time	Source	Destination	Protocol	Info
18:38:15.486609	192.168.1.15	192.168.1.10	UDP	Source port: 60104 Destination port: 55000
18:38:15.487221	192.168.1.10	192.168.1.200	TCP	56000 > 64477 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=978
18:38:15.710093	192.168.1.200	192.168.1.10	TCP	64477 > 56000 [ACK] Seq=1 Ack=979 Win=17520 Len=0
18:38:16.494712	192.168.1.15	192.168.1.10	UDP	Source port: 39080 Destination port: 55000
18:38:16.495316	192.168.1.10	192.168.1.200	TCP	56000 > 64477 [PSH, ACK] Seq=979 Ack=1 Win=64240 Len=977
18:38:16.714083	192.168.1.200	192.168.1.10	TCP	64477 > 56000 [ACK] Seq=1 Ack=1956 Win=16543 Len=0
18:38:17.493410	192.168.1.15	192.168.1.10	UDP	Source port: 39693 Destination port: 55000

FIGURE 13 - CAPTURE WIRESHARK DAEMON-COLLECTEUR-AFFICHEUR

10.2. SÉCURITÉ

La sécurité des communications n'est pas implémentée à cause des restrictions de temps. Néanmoins, la structure du système a été prévue pour qu'il soit relativement facile d'ajouter une couche SSL aux communications car Qt et Python le permettent.

10.3. MODULARITÉ

La séparation du superviseur en trois parties, l'utilisation de langages objet et l'utilisation de sockets de bas niveau, sont trois éléments qui permettent de le rendre extrêmement modulaire. Il est en effet envisageable qu'un autre afficheur soit créé pour afficher les données d'une autre manière (par exemple dans un navigateur web). Cet afficheur serait facilement intégrable au système existant pour autant qu'il respecte le protocole de communication.

Il peut y avoir plusieurs collecteurs, il faut configurer les daemons (via un fichier texte) pour envoyer les données à une liste de collecteurs.

Il peut également y avoir plusieurs afficheurs qui peuvent être sur la même machine qu'un collecteur ou sur une machine distante.

10.4. FILTRAGE

Dans la plupart des cas d'utilisation d'un système de monitoring, il n'est pas utile d'obtenir des renseignements sur les machines en bonne santé. Il s'avère donc nécessaire de filtrer les informations pour ne garder que celles qui ont un réel intérêt. Il faut donc choisir lequel des trois logiciels composant le superviseur va s'occuper de cette tâche.

Filtrer les informations dans le daemon a été envisagé car cela permet de répartir le travail et de diminuer l'utilisation de la bande passante en envoyant que les informations utiles. Cependant, c'est dans l'afficheur que le filtrage s'effectue car l'utilisation de la bande passante est négligeable. Ca permet ainsi de garder la modularité du système. En effet, il est envisageable d'avoir plusieurs afficheurs dont un qui n'effectuerait pas de filtrage.

11. DÉVELOPPEMENT ET STRUCTURE DU SUPERVISEUR

Cette section donne des informations générales sur le fonctionnement des différents logiciels composant le superviseur. Pour obtenir des informations plus techniques et plus détaillées, vous pouvez consulter la documentation fournie en annexe aux formats HTML et PDF.

11.1. DAEMON

Le daemon se charge de récupérer périodiquement les informations sur l'hyperviseur et ses machines virtuelles.
Une fois ces informations récupérées, il les transmet aux collecteurs qui ont été déclarés dans un fichier nommé "`collecteurs.conf.txt`". Ce fichier doit contenir une adresse de collecteur par ligne avec le format suivant: hôte[:port] .

Le daemon doit être exécuté sur chaque hyperviseur.

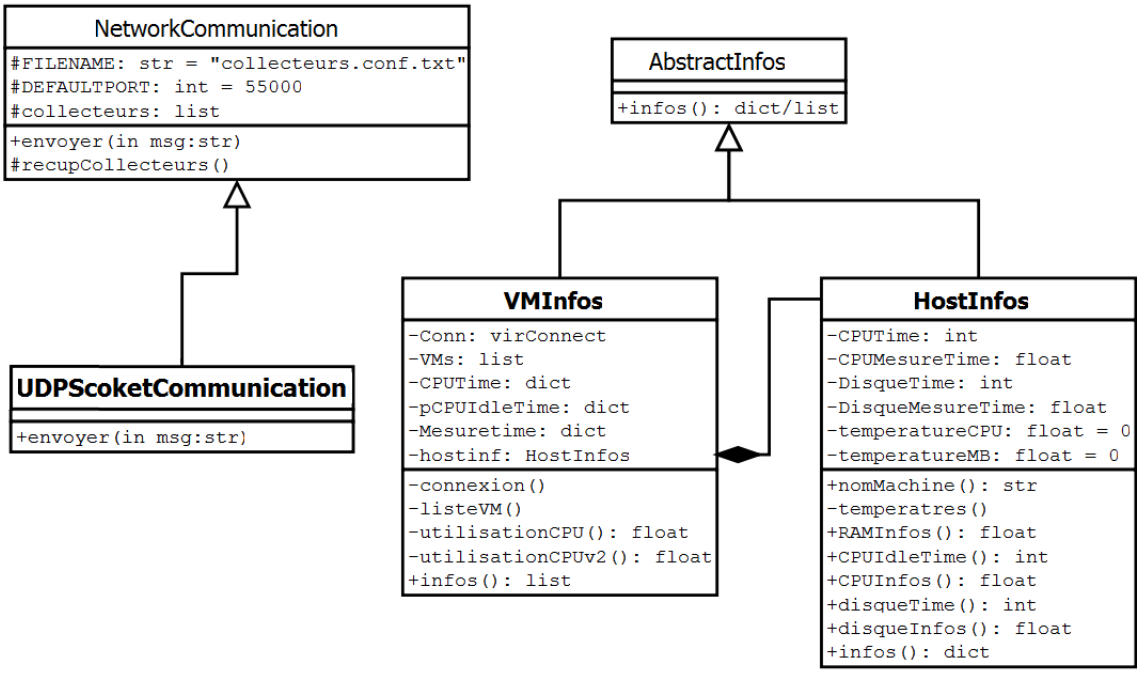


FIGURE 14 - DIAGRAMME DE CLASSES DU DAEMON

Il est composé d'une fonction `main()` qui est le point d'entrée et qui contient la boucle principale du programme. Cette fonction instancie les classes *UDPScoketCommunication, VMInfos et HostInfos*.

Il y a deux classes principales:

1. *NetworkCommunication* : cette classe va gérer l'envoi des données au collecteur.
2. *AbstractInfos*: cette classe va être utilisée pour récupérer les données.

La classe *NetworkCommunication* implémente une unique fonction qui lit le fichier "`collecteurs.conf.txt`" et stocke la liste des collecteurs en mémoire.

Toutes les classes qui héritent de *NetworkCommunication* doivent réimplémenter la méthode `envoyer(msg)` qui doit s'occuper d'envoyer `msg` aux collecteurs.

La classe *UDPScoketCommunication* qui hérite de *NetworkCommunication* contient le code nécessaire à l'envoi des informations via le protocole UDP. Si l'on souhaite utiliser le protocole TCP il faudrait créer une classe *TCPSocketCommunication* qui hériterait de *NetworkCommunication* .

Il y a deux classes qui héritent d'*AbstractInfos*:

1. *HostInfos*: cette classe va récupérer les informations sur l'hyperviseur
2. *VMInfos*: cette classe va récupérer les informations sur les machines virtuelles grâce à libvirt.

Toutes les classes qui héritent d'*AbstractInfos* doivent réimplémenter la méthode `infos()` qui s'occupe de fournir des informations sous la forme d'un dictionnaire ou d'une liste.

L'encodage des données au format JSON est effectué grâce à une fonction disponible dans la bibliothèque standard de Python.

Pour que le script Python puisse s'exécuter, il faut que le service *libvirtd* soit lancé et que les paquets suivants soient installés sur le système:

- python
- libvirt-python
- lm_sensors

Ces paquets peuvent être installés en exécutant cette commande: `yum install python libvirt-python lm_sensors`.

Pour l'exécuter, il faut ouvrir une console en tant que root, se placer dans le répertoire contenant le script, puis exécuter la commande suivante: `./daemon.py`. Il faut néanmoins que le fichier ait le droit d'exécution. Pour l'obtenir, vous pouvez exécuter la commande suivante: `chmod u+x,g+x,o+x daemon.py`.

11.2. COLLECTEUR

Le collecteur est le logiciel écrit en C++ avec le Framework multiplateforme Qt qui se charge de sauvegarder les données et de faire la liaison entre les daemons et les afficheurs.

La réception des données en provenance des daemons se fait via le port 55000. Les afficheurs se connectent au collecteur via le port 56000.

Le collecteur a quatre rôles qui correspondent chacun à une classe:

1. Récupérer les informations en provenance des daemons: classe *HyperviseurUDPSocket.*
2. Sauvegarder ces informations dans une base de données: classe *GestionBDD.*
3. Gérer les connexions TCP en provenance des afficheurs: classe *AfficheurSocket.*
4. Transmettre ces informations aux afficheurs actuellement connecté: classe *AfficheurSocketThread.*

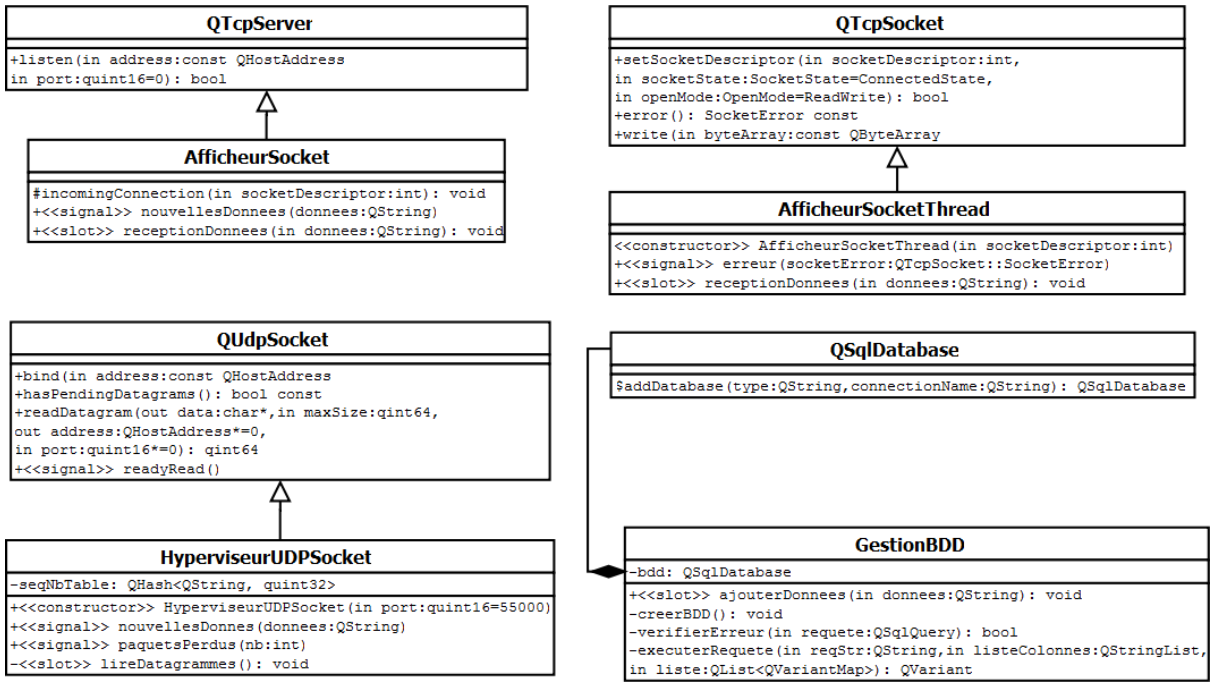


FIGURE 15 - DIAGRAMME DE CLASSES DU COLLECTEUR

Les données sont sauvegardées dans une base de données SQLite[29] qui réside dans la mémoire RAM pour des raisons de performance. Le contenu de cette base de données est régulièrement sauvegardé dans un fichier.

[29] Site officiel de SQLite : http://www.sqlite.org/

La base de données a la structure suivante:

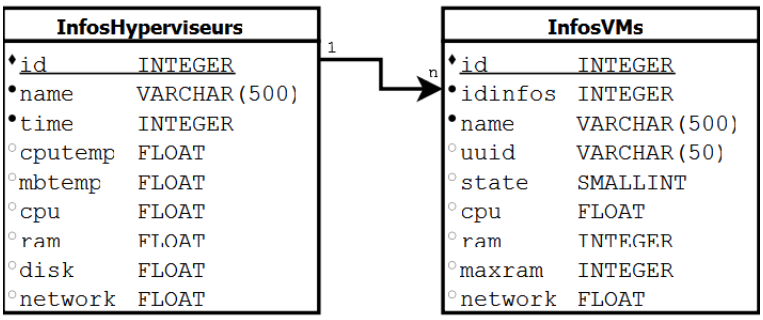


FIGURE 16 - SCHÉMA RELATIONNEL DE LA BASE DE DONNÉES DU COLLECTEUR

Qt ne propose pas de classe permettant de parser des chaînes JSON. Une bibliothèque externe a donc été utilisée. Cette dernière se nomme QJson et est spécialement prévue pour être utilisée dans des programmes faits avec Qt.

Le programme démarre dans la fonction `main()` du fichier `main.cpp` . Cette fonction va instancier un objet pour les trois premières classes, puis connecter le signal `nouvellesDonnes(QString donnees)` de l'instance de la classe *HyperviseurUDPSocket* au slot `ajouterDonnees(QString donnees)` de l'instance de la classe *GestionBDD* ainsi qu'au slot `receptionDonnees(QString donnees)` de la classe *AfficheurSocket*.

Le signal (au sens Qt) `nouvellesDonnes(QString donnees)` est émis lorsque de nouvelles données, en provenance d'un daemon, sont arrivées.

Lorsque que des données arrivent, l'instance de la classe *GestionBDD* va, dans un premier temps, parser ces données grâce à la bibliothèque *QJson* puis, dans un second temps, les ajouter à la base de données. Les données sur l'hyperviseur sont sauvegardées dans la table *InfosHyperviseurs* tandis que les données sur les machines virtuelles sont sauvegardées dans la table Infos VMs. Les informations sur les VMs sont reliées à l'hyperviseur grâce au champ *idinfos* qui correspond à l'id de l'entrée dans la table *InfosHyperviseurs.*

Lorsqu'un afficheur se connecte au collecteur, l'instance de la classe *AfficheurSocket* va créer une instance de la classe *AfficheurSocketThread*, qu'elle va ensuite déplacer dans un thread. Elle va également connecter le signal `nouvellesDonnes(QString donnees)` au slot `receptionDonnees(QString donnees)` de l'instance de la classe *AfficheurSocketThread* qui vient d'être créé. Pour chaque nouvelle connexion depuis un afficheur une instance de la classe *AfficheurSocketThread* est créée. Ces instances sont détruites lorsque l'afficheur se déconnecte.

11.3. AFFICHEUR

L'afficheur est composé de quatre classes:

1. *Afficheur* qui est la classe principale qui lie l'interface graphique et les autres classes.
2. *CollecteurSocket* qui se charge d'établir la connexion avec un collecteur et de transmettre les données à la classe *Afficheur*.
3. *ConnexionDialog* qui est la boite de dialogue demandant à l'utilisateur l'adresse et le port du collecteur.
4. *HyperviseurGraphique* qui correspond au graphique représentant l'hyperviseur.

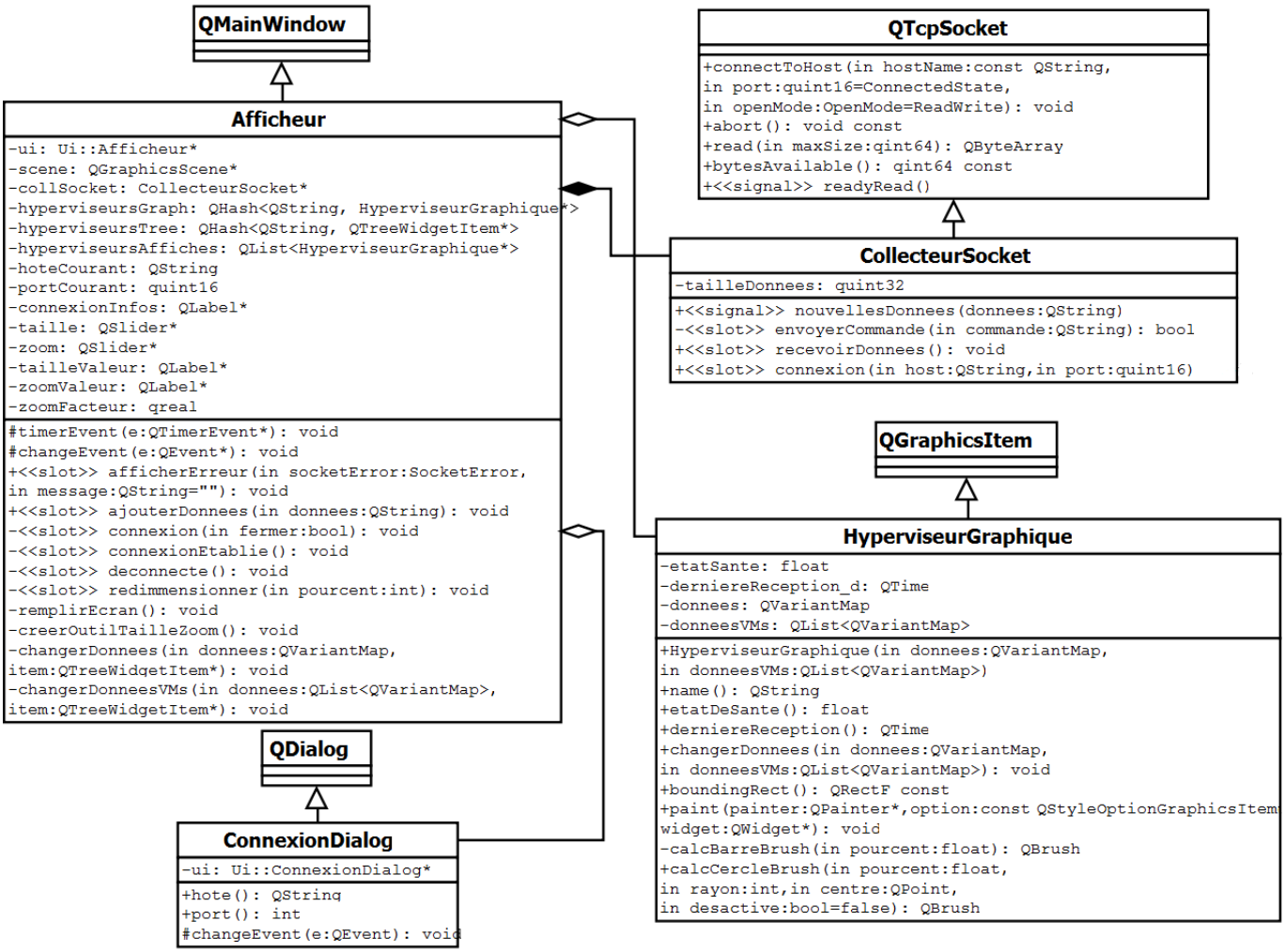


FIGURE 17 - DIAGRAMME DE CLASSES DE L'AFFICHEUR

Le logiciel commence son exécution dans la fonction `main()` située dans le fichier `main.cpp`. Cette fonction s'occupe uniquement d'instancier l'objet *Afficheur*.

L'afficheur est composé de deux parties principales. L'une qui contient une liste permettant de visualiser numériquement les informations sur les hyperviseurs et leurs machines virtuelles. L'autre qui contient une représentation graphique des données.

FIGURE 18 - GRAPHIQUE REPRÉSENTANT LE SUPERVISEUR ET SES MACHINES VIRTUELLES

Cette deuxième partie est l'élément clé de cet afficheur, elle permet de visualiser rapidement l'état de santé des hyperviseurs et de leurs VMs.

L'état de santé d'un hyperviseur est calculé en fonction du taux d'utilisation des ressources (CPU, RAM, …) et en fonction des températures du système. Il est représenté par la couleur de fond de la zone contenant le nom de l'hyperviseur. Plus cette couleur s'approche du rouge, plus l'état de santé devient critique.

Il est possible d'afficher seulement les machines dont l'état de santé est mauvais.

Pour chaque hyperviseur, une instance de la classe *HyperviseurGraphique* est créée et un nouveau graphique est ajouté dans la fenêtre. Un hyperviseur est supprimé lorsqu'aucune donnée n'a été reçue depuis trente secondes.

Sur le côté droit et sur le côté gauche, il y a des indicateurs de niveau qui donnent des informations sur les températures du système et sur la charge des différents composants matériels. Plus ces indicateurs sont élevés et plus le système s'approche d'un état critique. Leurs couleurs changent en fonction de leurs niveaux, elles varient entre le vert et le rouge.

Au centre sont représentées les machines virtuelles sous forme d'un cercle et d'un disque. La taille du cercle représente la taille de la mémoire RAM par rapport aux autres machines de l'hyperviseur. La taille du disque représente la quantité de mémoire RAM utilisée. La couleur du disque représente la charge CPU de la VM, plus la couleur s'approche du rouge, plus la charge est élevée. Les VMs qui ne sont pas en cours d'exécution sont représentées par un disque gris.

12. TESTS EFFECTUÉS

Tout au long du développement du superviseur, des tests ont été effectués pour valider les différentes parties du programme.

12.1. DAEMON

TEST DE LA CONNEXION À LIBVIRT VIA UN SCRIPT PYTHON

Pour effectuer ce test, une connexion a été établie grâce à l'API libvirt. Des données sur les machines virtuelles (nom, état, etc.) ont été imprimées sur la console pour vérifier le fonctionnement du code.

TEST DE L'OUVERTURE D'UN FICHIER ET D'UN PROCESSUS DEPUIS UN SCRIPT PYTHON

Ce test a été effectué en ouvrant un fichier avec la fonction `fopen` et en ouvrant un processus avec la fonction `popen`. Le contenu du fichier et le résultat du processus ont ensuite été imprimés sur la console pour pouvoir vérifier le bon fonctionnement de ces deux commandes.

TEST DES VALEURS OBTENUES

Les valeurs obtenues grâce à libvirt et grâce aux informations données par */proc* ont été comparées aux valeurs indiquées par les outils disponibles sur le système hôte et sur les machines virtuelles.

TEST DE LA CONVERSION AU FORMAT JSON DANS LE SCRIPT PYTHON

La structure obtenue a été convertie au format JSON grâce à la fonction disponible dans la bibliothèque standard de Python. Le résultat a ensuite été imprimé sur la console puis comparé avec la chaîne JSON créée manuellement lors de la spécification.

TEST DE L'ENVOI DES DONNÉES VIA UDP EN PYTHON

Les données, une fois converties au format JSON, ont été envoyées en UDP à une autre machine. Le logiciel Wireshark était lancé sur cette dernière et capturait les paquets arrivant sur la carte réseau. Les paquets ont été analysés à l'aide de ce logiciel afin de déterminer s'ils correspondaient à la spécification.

12.2. COLLECTEUR

TEST DE LA RÉCEPTION DES DONNÉES EN UDP

Un code minimal chargé de recevoir les données en provenance du collecteur à été réalisé. Les données ont ensuite été affichées sur la console afin de vérifier qu'elles correspondent à celles envoyées par le daemon.

TEST DU PARSING DES DONNÉES JSON

La bibliothèque QJson a été compilée puis utilisée dans le code du collecteur pour parser les données en provenance du daemon. Le débogueur a ensuite été utilisé pour vérifier si la structure de donnée correspond à celle présente dans le daemon.

TEST DE LA BASE DE DONNÉES (CRÉATION ET INSERTION)

La base de données a été créée dans le collecteur. Le fichier contenant la base de données a été ouvert grâce à un programme nommé "SQLite Browser". Ceci a permis de vérifier que la base de données a été créée correctement. Les données en provenance du daemon ont ensuite été ajoutées à la base de données. Le résultat a été vérifié en utilisant le logiciel "SQLite Browser".

TEST DU SERVEUR TCP

Le serveur TCP a été testé en utilisant le client Telnet. En effet un client Telnet est capable de créer une session TCP.

TEST DE L'ENVOI DES DONNÉES VIA LA CONNEXION TCP

Le test de l'envoi des données a également été effectué avec le client Telnet. Les paquets ont été capturés avec Wireshark pour vérifier leur contenu.

12.3. AFFICHEUR

TEST DU DESSIN DU GRAPHIQUE REPRÉSENTANT UN HYPERVISEUR.

Le graphique généré par le programme a été comparé au prototype qui a été réalisé plus tôt grâce à Photoshop.

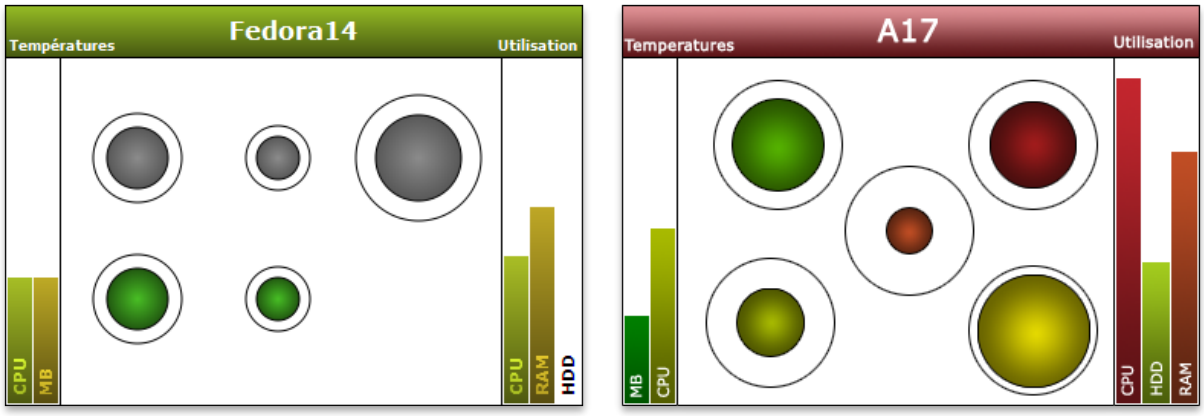


FIGURE 19 - GRAPHIQUE GÉNÉRÉ (À GAUCHE) ET LE PROTOTYPE (À DROITE)

TEST DE L'ALGORITHME DE PLACEMENT DES VMS SUR LE GRAPHIQUE

Un grand nombre de machines virtuelles ont été créées pour vérifier le bon fonctionnement de l'algorithme de dimensionnement et de placement des cercles représentant les VMs.

FIGURE 20 - TEST DE L'ALGORITHME DE PLACEMENT DES VMS

TEST DE LA CONNEXION TCP ET DE LA BONNE RÉCEPTION DES DONNÉES

Ce test a été effectué en comparant les données numériques affichées dans la liste des hyperviseurs et des VMs aux données affichées directement sur les hyperviseurs et sur les VMs.

TEST DU PLACEMENT DES GRAPHIQUES REPRÉSENTANT LES HYPERVISEURS

Ce test a été effectué en ajoutant des hyperviseurs, puis en modifiant les dimensions de la fenêtre pour vérifier que le positionnement se comportait correctement.

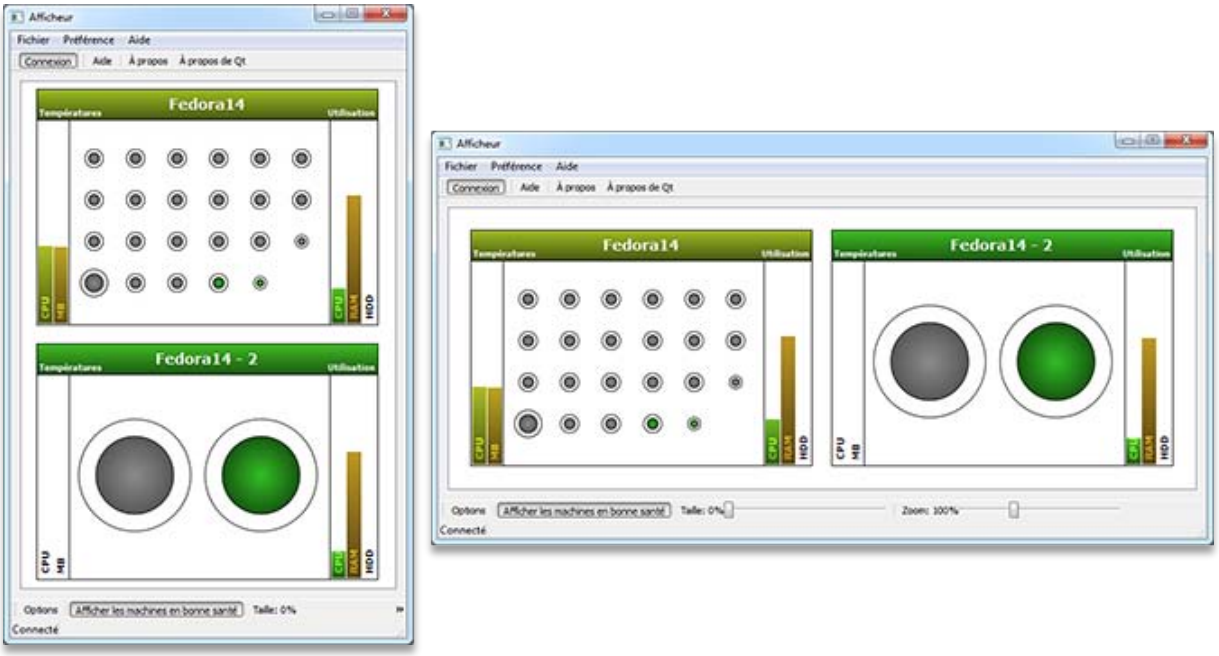


FIGURE 21 - TEST DU PLACEMENT DES GRAPHIQUES REPRÉSENTANT LES HYPERVISEURS

12.4. TESTS GÉNÉRAUX

Une fois que tous les tests précédents ont été effectués, des tests plus généraux mettant en œuvre les trois logiciels du superviseur on été réalisés.

TEST DE LA STABILITÉ DU SUPERVISEUR

Les trois logiciels composant le superviseur ont été exécutés pendant une durée supérieure à trois jours. Après ces trois jours aucun problème n'est apparu et le système continuait à fonctionner correctement.

MISE EN PRATIQUE DES SCÉNARIOS

Les scénarios ont tous été mis en pratique pour effectuer une validation finale du superviseur.

FIGURE 22 - GÉNÉRATION D'UNE CHARGE CPU DE 60% SUR UNE VM

13. PROBLÈMES RENCONTRÉS

13.1. CRÉATION D'UN BRIDGE RÉSEAU[30]

J'ai rencontré un problème lorsque j'ai créé un bridge avec virt-manager pour connecter les machines virtuelles sur le réseau physique. Après avoir validé les paramètres, la configuration prenait beaucoup de temps. Après cinq minutes j'ai annulé l'opération, ce qui a eu pour effet de geler le logiciel. J'ai donc dû le tuer manuellement. Lorsque j'ai voulu recommencer la procédure, le bridge n'apparaissait pas mais était néanmoins dans les fichiers de configurations, ce qui m'empêchait de le recréer. J'ai donc dû supprimer manuellement les modifications qui avaient été apportées au système.

13.2. PARE-FEU LINUX

Le pare-feu du système Linux pose également quelques problèmes. Par exemple lorsque les machines virtuelles sont configurées en DHCP, la communication avec le serveur DHCP ne fonctionne pas et par conséquent la machine n'arrive pas à récupérer les paramètres réseau. Comme la configuration du pare-feu est fastidieuse et ne fait pas partie de ce projet, j'ai décidé, en accord avec mon professeur, de le désactiver. Pour cela, il faut exécuter la commande suivante: `service iptables stop`. Si vous souhaitez le désactiver de manière permanente, il faut exécuter cette commande (sous Fedora uniquement): `chkconfig –del iptables`.

13.3. ENDIANNESS

Lorsque j'ai développé le collecteur, le numéro de séquence qui était reçu ne correspondait pas à celui qui était envoyé par le daemon. Après avoir effectué des acquisitions avec Wireshark, j'ai constaté que le problème venait de la représentation des données. Le daemon envoyait le nombre en little-endian alors que le collecteur l'interprétait comme étant en big-endian. J'ai donc spécifié explicitement des deux côtés d'utiliser le format big-endian qui correspond au standard *network byte order* défini dans le protocole IP.

13.4. RÉSEAU WIFI

En testant l'afficheur sur mon ordinateur portable tournant sous Windows 7, j'ai remarqué un décalage entre le moment où le collecteur envoyait les données et le moment où l'afficheur les dessinait. J'ai effectué plusieurs tests ainsi que plusieurs optimisations de la connexion et de l'afficheur sans comprendre le problème. J'ai ensuite compilé l'afficheur sur un ordinateur du laboratoire tournant sur Fedora. Sur cet ordinateur tout fonctionnait. À partir de ce moment, il n'y avait plus que deux possibilités: un problème avec le système d'exploitation ou un problème avec la connexion réseau. Le problème venait en réalité d'un dispositif qui retarde la distribution des données sur le réseau wifi.

[30] Le bug sur le bug-tracker de Red Hat: https://bugzilla.redhat.com/show_bug.cgi?id=709734
Annexe A.6 : Utilisation des mailing-list et des bug-tracker

14. CONCLUSION

Les objectifs de ce travail qui s'est articulé sur deux grandes parties, l'étude de Linux et de KVM ainsi que la réalisation du superviseur, ont tous été atteints. Bien que les logiciels développés puissent encore être améliorés et optimisés, le côté fonctionnel a été obtenu.

Ce travail m'a permis de travailler avec de nouvelles technologies qui, malgré leur jeune âge, sont robustes et pleinement fonctionnelles. Cette rapidité et cette qualité de développement sont, sans nul doute, due aux caractéristiques libres et open sources qui permettent à toute personne de participer au développement de ces technologies.

Ce projet m'a également permis de développer une suite de logiciels qui peuvent communiquer entre eux en franchisant les barrières du langage de programmation et du système d'exploitation.

Les logiciels que j'ai réalisé m'ont permis de mettre en pratique et de valider les notions théoriques sur les réseaux IP apprises durant mes trois années à l'HEPIA.

Ce travail fut également ma première expérience académique d'un projet réalisé à temps plein sur huit semaines consécutives. En effet, une semaine académique standard est partagée entre plusieurs cours et plusieurs projets et travaux pratiques. Cette expérience fut donc très enrichissante.

A. ANNEXES

A.1. MATÉRIEL À DISPOSITION

Le laboratoire de transmission de données de l'HEPIA possède plusieurs ordinateurs compatibles avec les technologies de virtualisation telles que Linux-KVM, VMware ESX(i), Intel VT, etc. Il dispose également d'un intranet qui relie ces machines entre elles et sur internet.

Ces machines ont pu être utilisées pour simuler une grille d'hyperviseur. La distribution Fedora a été installée sur ces machines avec les outils permettant de faire de la virtualisation grâce à Linux-KVM.

La configuration hardware de ces machines est la suivante:

- Carte mère ASUS P5Q-VM DO uATX (Intel Q45/ICH10DO - Socket 775 - FSB 1333)
- CPU Intel Core 2 Duo E8400 3.0GHz
- RAM 4Go DDR2
- Chpiset Intel® Q45 / ICH10DO support Intel Active Management Technology
- LAN Intel 82567LM Gigabit LAN Phy controller

J'ai également utilisé mes ordinateurs personnels (fixe et portable).

A.2. LE PSEUDO SYSTÈME DE FICHIER *PROC*

Le système de fichier *proc* [31]est un pseudo système de fichiers sur les systèmes Linux qui permet d'interroger et de configurer le noyau. Sur la plupart des distributions de Linux, il est monté à l'emplacement */proc*.

La racine de proc contient une multitude de fichiers et de dossiers. Les fichiers à la racine concernent le système en général alors que les dossiers contiennent des fichiers qui concernent un processus particulier. Il y a un dossier par processus et son nom correspond au PID de ce processus.

Il est, par conséquent, très facile d'obtenir des informations du noyau (telles que l'utilisation des périphériques, des renseignements sur un processus, etc.) car il suffit de lire un fichier. De même pour paramétrer le noyau, il suffit d'écrire dans un fichier.

A.3. INSTALLATION D'UNE MACHINE VIRTUELLE

L'installation d'une machine virtuelle peut se faire de plusieurs manières. Il est possible d'utiliser virt-manager ou d'utiliser la ligne de commande. Il est également possible de l'installer directement en spécifiant l'adresse web pointant sur l'image d'un système.

Les procédures d'installation énumérées ci-dessus sont expliquées en détail dans le document de M. Pasche dans le chapitre 8 "Création de machines virtuelles" (page 81). Lien direct: http://www.tdeig.ch/kvm/pasche_R.pdf#page=97

[31] Page de manuel de *proc* sur kernel.org : http://www.kernel.org/doc/man-pages/online/pages/man5/proc.5.html

A.4. CRÉATION D'UN BRIDGE RÉSEAU

Lorsque l'on crée une machine virtuelle avec *virt-manager,* une carte réseau connectée au réseau virtuel *default* (configuré en mode NAT isolé) est ajoutée par défaut. Ce réseau *default* est isolé: les machines virtuelles connectées à ce réseau peuvent communiquer entre elles mais elles ne peuvent pas communiquer avec l'extérieur.

Le mode NAT permet de faire cohabiter toutes les machines virtuelles dans un réseau qui leur est dédié. Ce mode ajoute un serveur DHCP dans ce réseau pour permettre l'attribution automatique des paramètres réseau aux différentes VMs.

Si l'on souhaite connecter nos machines virtuelles directement au réseau qui est derrière une carte réseau physique, il faut utiliser un bridge pour les lier à cette dernière.

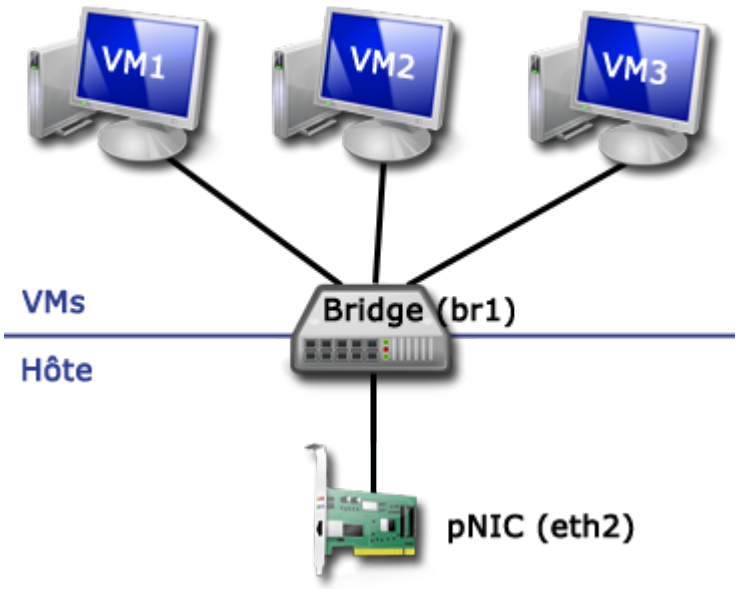


FIGURE 23 - CONNEXION DES VMs À UNE pNIC VIA UN BRIDGE

Il existe deux méthodes pour mettre en place cette configuration réseau.

1. Via une interface graphique (*virt-manager*).
2. Via la ligne de commande (*virsh iface-define*).

[32]La configuration d'interfaces réseaux via libvirt n'est pas encore totalement au point. En effet, il faut plusieurs minutes pour que la création de l'interface s'accomplisse et l'interface ne redémarrera pas avec la machine même si l'option "onboot" a été spécifiée.

[32] Le bug sur le bug-tracker de Red Hat: https://bugzilla.redhat.com/show_bug.cgi?id=709734
Annexe A.6 : Utilisation des mailing-list et des bug-tracker

A.4.1. CONFIGURATION GUI

Pour installer le bridge via cette méthode, il faut lancer `virt-manager`, puis aller sous "Édition -> Détails de la connexion -> Interfaces réseau". Arrivé ici, il suffit de cliquer sur l'icône "+" en bas à gauche.

Dans l'assistant de création, il faut sélectionner "Pont -> Suivant". Dans la fenêtre qui suit, il faut sélectionner la carte réseau sur laquelle on veut relier le bridge, cocher la case "Activer maintenant" et sélectionner "onboot" comme mode de démarrage. Si vous souhaitez définir manuellement les paramètres ip, il faut cliquer sur le bouton configurer. Vous pouvez ensuite valider en cliquant sur "Terminer".

Une fois ces étapes effectuées, la carte réseau est associée au bridge qui vient d'être créé. Il faut maintenant relier les machines virtuelles au bridge en ouvrant la machine virtuelle puis en cliquant sur "Afficher -> Détails" et en changeant le périphérique source de la carte réseau.

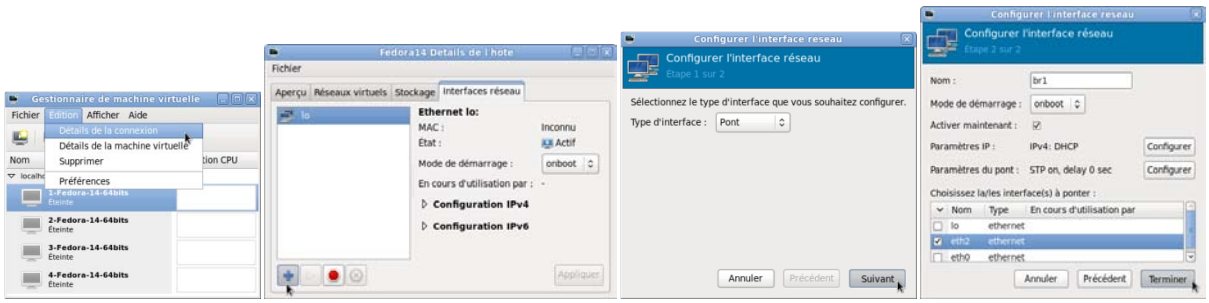


FIGURE 24 - CONFIGURATION D'UN BRIDGE AVEC VIRT-MANAGER

A.4.2. CONFIGURATION CLI

Si l'on souhaite utiliser le Shell libvirt (`virsh`), il faut tout d'abord créer un fichier XML qui va contenir les paramètres de notre bridge.

```
<interface type="bridge" name="br1">
      <start mode="onboot"/>
      <protocol family="ipv4">
            <dhcp/>
            <route gateway='192.168.1.1'/>
      </protocol>
      <protocol family="ipv6">
            <ip address="fe80::21b:21ff:fe43:47a1" prefix="64"/>
      </protocol>
      <bridge stp="off" delay="0.01">
            <interface type="ethernet" name="eth2">
                  <mac address="00:1b:21:43:36:b0"/>
            </interface>
      </bridge>
</interface>
```

FICHIER BRIDGEINTERFACECONFIG.XML – CONFIGURATION D'UN BRIDGE

Dès que ce fichier est créé, il faut exécuter cette commande : `virsh iface-define bridgeInterfaceConfig.xml`.
Une fois cette dernière terminée, il faut activer l'interface avec la commande suivante: `virsh iface-start br1` (attention, il faut ~10mn pour que la configuration s'effectue).

Ensuite, il faut aller modifier les fichiers de configuration du domaine grâce à la commande `virsh edit <domaine>`.

A.5. INSTALLATION DE IOMETER ET DE FIO SOUS LINUX

Fio[33] est disponible dans les dépôts de la plupart des distributions. Sous Fedora il faut exécuter la commande `yum install fio` pour l'installer.

A.5.1. FIO

Pour lancer un benckmark, vous pouvez lancer la commande suivante: `fio -name iops -rw=randrw -runtime=30 -filename /dev/sda`. Cette commande effectue un benchmark en lecture et en écriture. Si le disque *sda* contient des partitions elles seront endommagées. Si vous ne souhaitez pas endommager vos partitions, ne faites pas de benchmark en écriture.

A.5.2. IOMETER

L'installation d'Iometer sur des systèmes Linux est un peu plus compliquée que celle de fio. En effet, il n'existe pas de paquet dans les dépôts et les versions précompilées ne fonctionnent pas sur la plupart des systèmes. Il faut donc compiler Iometer depuis les sources.

Les sources sont téléchargeables à cette adresse: http://sourceforge.net/projects/iometer/files/iometer-stable/2006-07-27/iometer-2006_07_27.common-src.zip/download

Il faut décompresser l'archive et renommer le fichier makefile de la version de votre système en "Makefile" (ex: "Makefile-Linux.x86_64" -> "Makefile")

Dans la version actuelle de Iometer (2006_07_27) une erreur se produira à la compilation car les distributions récentes de Linux ne fournissent plus le fichier "`stropts.h`" qui est inclus dans `IOPerformance.h`. Ce fichier est en réalité utile que sur les systèmes Solaris. Il faut donc remplacer la ligne `"#if defined(IOMTR_OS_LINUX) || defined(IOMTR_OSFAMILY_NETWARE) || defined(IOMTR_OS_SOLARIS)"` (ligne 100) par la ligne `"#if defined(IOMTR_OSFAMILY_NETWARE) || defined(IOMTR_OS_SOLARIS)"`. Vous pouvez ensuite lancer la commande "`make dynamo`" et la compilation devrait fonctionner (test effectué sur une distribution Fedora 14).

L'interface graphique est disponible uniquement sur Windows, il faut donc installer et lancer Iometer sur un système Windows puis lancer `dynamo` sur le système Linux (commande: "`./dynamo -i <IOmeter_IP> -m <IP>`" *Iometer_IP* est l'IP de la machine Windows et *IP* correspond à l'IP de la machine linux). Le PC Linux sera visible dans Iometer (sur le PC Windows).

Il est préférable de désactiver *iptables* (le pare-feu de Linux) pendant l'exécution de `dynamo` pour éviter des problèmes de connexion dûs au pare-feu "`service iptables stop`".

[33] Page de manuel de fio : http://linux.die.net/man/1/fio

A.6. UTILISATION DES MAILING-LIST ET DES BUG-TRACKER

Lorsque l'on travaille avec des logiciels libres et que l'on découvre un problème ou que l'on souhaite proposer une amélioration, il est courant d'utiliser soit des mailing-list, soit des bug-tracker.

A.6.1. MAILING-LIST

Une mailing-list est une liste de distribution à laquelle il faut s'abonner pour recevoir les réponses.
Les principaux développeurs sont tous inscrits à cette mailing-list et reçoivent les demandes directement dans leur boite e-mail.

Les mailing-list sont en général très actives.

Le point négatif de ce système est la recherche des bugs déjà postés.
En effet, en s'inscrivant à la mailing-list, on ne reçoit pas les messages antérieurs à la date d'inscription, il est donc difficile d'effectuer une recherche.
Pour pallier à ce problème, il existe sur des sites web des archives des mails échangés.

A.6.2. BUG-TRACKER

Un bug-tracker est, en général, une application web qui se présente sous la forme d'un site web et permet de poster des bugs et des idées d'améliorations.

Ce système gère une base de données des bugs et permet de vérifier si le bug que l'on souhaite signaler n'est pas déjà présent dans la base de données.

Il est possible de suivre un bug pour être notifié lors d'un changement. Ceci permet de choisir les bugs ou les catégories de bugs desquels on veut recevoir des notifications par e-mail.

A.7. POSSIBILITÉS D'AMÉLIORATION

Vous trouverez ci-dessous une liste des possibilités d'améliorations qui n'ont pas été implémentées à cause des restrictions de temps.

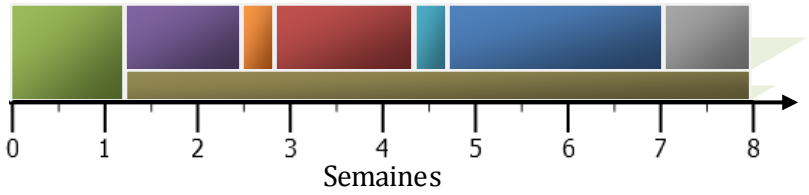
- Ajouter une couche SSL aux communications réseau.
- Surveiller l'utilisation du réseau (cette fonction est déjà implémentée dans le collecteur).
- Utiliser les patches[34] permettant de récupérer la vraie valeur de l'utilisation de la mémoire RAM des VM via le driver de ballooning.
- Permettre à l'afficheur d'envoyer des commandes au collecteur pour ne récupérer qu'une partie des données ou des données plus anciennes.
- Comme le daemon utilise libvirt pour récupérer les informations sur les machines virtuelles, il serait envisageable de l'adapter pour superviser d'autres hyperviseurs tels que VMware ESX(i), Xen, etc.

[34] Archive de la mailing-list (cliquez sur *Date Prev* pour voir les patches):
http://www.redhat.com/archives/libvir-list/2009-December/msg00618.html

A.8. ORGANISATION DU TEMPS

Ce travail de diplôme s'est déroulé sur huit semaines à temps plein.

Le chronogramme ci-dessous détaille la répartition du travail sur les huit semaines.



- Prise en main de Fedora et des outils de virtualisation.
- Rédaction du mémoire.
- Étude du travail de M. Sébastien Pasche et étude des outils de mesure de performances.
- Mise au point des scénarios.
- Développement des outils de générations de charge.
- Élaboration de l'architecture du superviseur.
- Développement des logiciels composant le superviseur.
- Mise en pratique des scénarios, génération de la documentation.

A.9. COMMANDES UTILES

Lors de l'étude des outils Linux en relation avec les mesures de performances, certains outils ou fonctions ont été découverts. Certains de ces derniers n'étaient pas utiles à ce travail, néanmoins il serait dommage de perdre ces informations. C'est pourquoi ils ont été répertoriés ci-dessous.

- Voir les driver Virtio : `modprobe –l | grep virtio`
- SysRq : http://fr.wikipedia.org/wiki/Magic_SysRq_key
  Afficher la liste des différentes tâches actives ainsi que des informations pour chacune : `echo t > /proc/sysrq-trigger`
  Puis lire le contenu du fichier dump : `dmesg` ou `cat /var/log/messages`
- Surveiller les appels système et les signaux : `strace –T`
- Surveiller les appels IO : `blktrace`
- Gérer les ponts (bridges): `brctl`
- Désactiver le pare-feu: `chkconfig --del iptables`
- Changer l'ordonnanceur IO : http://www.mjmwired.net/kernel/Documentation/block/switching-sched.txt et http://en.wikipedia.org/wiki/I/O_scheduling
- Lancer une application avec une priorité IO : http://manpages.ubuntu.com/manpages/intrepid/fr/man1/ionice.1.html
- Utiliser FIO (Flexible I/O tester) : `fio -name iops -rw=randrw -runtime=30 -filename /dev/sda`
- Connaître l'association LV<->DM-x : `lvdisplay` (ligne Block device après le caractère :)
- Monter un disque d'une VM dans l'hôte (expérimental) : `guestfish --ro -d "[VM_name]" –i`
- Accéder à la console d'une VM (si elle a un port série connecté à une console): `virsh console "[VM_name]"`
- Statistiques détaillées des disques: `iostat -x 3` (rafraichissement toutes les 3 secondes). Attention: ne pas prendre en compte les premières données.
- Suivre l'évolution du résultat d'une commande : watch -d --interval=1 <commande>

B. TABLE DES ILLUSTRATIONS

C. ANNEXES EXTERNES

Vous trouverez ci-joint à ce document les annexes suivantes:

- Aide des logiciels de génération de charge.
- Documentation du daemon.
- Documentation du collecteur.
- Documentation de l'afficheur.
- CD-ROM contenant:
  - Le mémoire au format PDF.
  - L'aide des logiciels de génération de charge au format PDF.
  - Les sources et les exécutables (Linux et/ou Windows) des trois logiciels composants le superviseur.
  - Les sources et les exécutables (Linux) des logiciels de génération de charge.
  - Les documentations (aux formats PDF et HTML) des trois logiciels composants le superviseur.