Une architecture de workspaces ubiquitaires sécurisée et distribuée

Jonathan Rouzaud-Cornabas

Laboratoire d'Informatique Fondamentale d'Orléans
Bâtiment IIIA, Rue Léonard de Vinci
45067 Orléans, France
jonathan.rouzaud-cornabas@univ-orleans.fr

Résumé Les architectures supportant des environnements collaboratifs apportent de nombreux problèmes de sécurité. Un de ces environnements est l'infrastructure de bureaux distants qui fournissent un environnement graphique distant à travers le réseau à des clients légers. Comme un grand nombre de clients légers ont accès aux mêmes machines, les risques de conflits et d'interférences doivent être soigneusement étudiées. Même s'il existe de nombreuses solutions permettant d'améliorer la sécurité d'une machine, aucune ne propose une vraie solution permettant de sécuriser une architecture de bureaux graphiques distants complète.

Dans ce papier, nous étendons un travail précédemment réalisé [2] où nous avions décrit une architecture avec une authentification lourde celle-ci supporte le Single Sign On et supportant un ensemble d'outils de détections d'intrusions et d'anomalies. Et particulièrement une vérification du comportement des utilisateurs via un système de corrélation. De plus, des aspects de Load Balancing via l'introduction d'un algorithme de répartition se basant sur l'utilisation des ressources de chaque machine avaient été introduit.

Les extensions portent sur une meilleur virtualisation et donc une meilleure gestion des problèmes de conflits et d'interférences entre les utilisateurs, une meilleure répartition de la charge des bureaux graphiques et la combinaison de nouveaux concepts de sécurité. Cela nous permet d'améliorer la disponibilité, la qualité de services, la reprise sur panne et la sécurité globale de l'architecture que nous proposons.

1 Introduction

De plus en plus de personnes disposent d'un ordinateur mais le niveau moyen de connaissance en informatique des utilisateurs ne leur permet pas, en général, d'avoir une machine sécurisée que ce soit au niveau de l'intégrité de leurs données ou de la pérennité du fonctionnement de leur système. Nous avons également constaté qu'un grand nombre des virus et attaques actuelles se basent sur le principe que les utilisateurs n'ont pas ou peu de connaissances sur comment sécuriser leur ordinateur et comment identifier une menace. Enfin, un comportement émergeant des dernières années, est le nomadisme des utilisateurs ce qui amène souvent des problèmes d'accès à distance aux données, de disponibilités des ressources et de sécurité des données et de l'infrastructure (terminaux et serveurs).

Devant ce constat faisant ressortir les difficultés pour l'utilisateur d'utiliser des logiciels de sécurité toujours plus nombreux et complexes mais aussi, devant un besoin de sécurité constamment renforcé que ce soit pour une utilisation sédentaire ou nomade, nous avons souhaité mettre en place une structure permettant depuis n'importe où et sur un grand nombre de dispositifs (PC, PDA, SmartPhone, etc.) d'avoir un bureau graphique complet et sécurisé. Pour cela, nous avons implanté et mis en place les principes présentés dans [2]. Cette architecture a permis de répondre à un

grand nombre de nos attentes au niveau conceptuel. Mais de nombreux points restés à améliorer tels que la virtualisation et la répartition de la demande de ressources par les utilisateurs sur un ensemble d'ordinateur. En effet, nous nous reposons sur un système entièrement centralisé où un grand nombre d'utilisateurs potentiellement concurrents vont évoluer de concert. Nous souhaitions donc renforcer les aspects de virtualisation en intégrant une meilleure sécurité. Nous souhaitons également une gestion plus fine des ressources utilisées par chaque utilisateur nous permettrait de mettre en place une qualité de service.

De plus, avec l'augmentation de la bande passante avec ou sans fil et la puissance des dispositifs légers portables, les bureaux distants sont devenus une alternative envisageables dans des contextes de PME voire même d'administration.

2 Etat de l'art

2.1 Solutions actuelles de Remote Desktop

Dans le monde des logiciels propriétaires, deux solutions principales de workspaces ubiquitaires (i.e. des bureaux graphiques dans notre cas) cohabitent :

- Sun avec Sun Ray (voir [4]) comprend aussi bien un système d'exploitation (Solaris) que des utilitaires et un client léger. Le problème de cette solution est qu'elle est entièrement propriétaire et n'est clairement pas orientée vers le service à l'utilisateur mais plus une orientation accès Mainframe (Console). La répartition à chaud des workspaces n'est pas possible et SunRay ne dispose pas de dispositif d'isolement complet.
- L'autre solution propriétaire qui est également la solution qui est la plus utilisée est une combinaison de produits venant de trois sociétés : VMWare, Microsoft et Citrix. VMWare fournissant la virtualisation (lourde) et la répartition de charges des machines virtuelles sur un ensemble de machines hôtes, Citrix le protocole d'accès aux bureaux distants et Microsoft le système d'exploitation. Mais VMWare apporte une sur-utilisation importante et les solutions de sécurité sous MS-Windows sont bien moins avancées et l'OS beaucoup plus fragile que les autres solutions.

Dans le monde de l'open source, de nombreuses solutions, plus ou moins, complètes coexistent. Xen et Globus Workspace (voir [3]) permettent de répartir un ensemble de machines virtuelles (virtualisation lourde) sur un ensemble de machines hôte mais aucun mécanisme de gestion de bureau distant n'y est incorporé, le but étant plus la répartition de ressources pour des services web sur une grille que la mise en place de bureaux graphiques distants. Ulteo est une distribution ayant également pour but de fournir un bureau graphique distant mais les avancements sont faibles et l'architecture actuelle pose de gros problèmes de sécurité et d'utilisation de ressources, de par sa conception, pour être fiable lors d'un passage à une échelle réelle.

Finalement, les bureaux distants "Web2.0" qui ne sont ni plus ni moins que des pseudo-bureaux graphiques en AJAX simulent un bureau distant en utilisant un mélange de procédure stockée coté serveur et de code javascript coté client. Ce ne sont pas réellement des bureaux distants complets et ne nous y intéresserons pas dans le reste de l'article, dans la mesure où les fonctionnalités offertes restent très limitée. Le problème principale étant le besoin de ré-écrire toutes les applications depuis le début pour pouvoir les proposer sur de tels "bureaux".

2.2 Pourquoi laisser chroot pour OpenVZ

Précédemment dans [2], nous avions choisi de nous orienter pour une virtualisation légère en raison de sa faible sur-utilisation dû à son incorporation dans notre architecture, mais également, car l'ensemble des machines virtuelles a une base strictement identique. Pour simuler cette virtualisation légère, nous avions développé et mis en place un module de construction automatique d'environnement virtuel pour l'utilisateur basé sur le système d'authentification PAM et la fonction système chroot.

Dans le cadre d'une première étude, chroot nous a permis d'évaluer nos concepts et nos choix. Mais nous nous sommes rapidement rendus compte que même en augmentant la sécurité des chroot via des patchs tel que GRSecurity, nous ne pourrions pas avoir un réel environnement virtualisé en utilisant chroot, même en l'étendant. Nous nous sommes donc orientés vers OpenVZ, une solution de virtualisation légère ayant les fonctionnalités que nous avions déjà via l'utilisation de chroot et en ajoutant de nouvelles qui nous ont permis d'étendre nos fonctionnalités.

Contrairement à chroot, OpenVZ (voir [5]) ne protége pas uniquement les accès de l'utilisateur sur le système au niveau du système de fichier mais de l'ensemble du système. L'utilisateur dispose donc de ses propres périphériques, signaux systèmes et autres fonctionnalités systèmes. Il ne pourra pas "s'évader" simplement comme il est possible via chroot.

La contre-partie est un patch noyau apportant des modifications permettant d'utiliser le kernel GNU/Linux comme un hyperviseur de machine virtuelle légère. Ce patch noyau enlève la compatibilité avec SELinux et GRSecurity ce qui nous empêche temporairement de bénéficier d'un système de contrôle d'accès MAC.

2.3 Live Migration

Globus Workspace est un projet opensource visant à répartir la charge d'un ensemble de machine Xen sur une grille. Leur méthode repose sur les limitations d'utilisation de ressources de chaque machine virtuelle et la disponibilité de ces dernières. Il se base donc sur des données statiques afin de pouvoir répartir selon le meilleur des cas un ensemble de machines virtuelles. VMWare propose des produits permettant de répartir la charge entre plusieurs machines. Les produits étant propriétaire et fermé, nous ne connaissons pas la nature de leur implémentation et quel mécanisme ils utilisent.

3 Motivation

3.1 Virtualisation

Dans l'architecture présentée dans [2], nous avions expliqué que nous ne souhaitions pas utiliser une virtualisation lourde car elle entraînait une sur-utilisation massive des ressources. De plus, l'ensemble des machines virtuelles étant identique, nous pouvions avoir un bien meilleur compromis en utilisant une virtualisation légère. En partant de ces faits, nous nous sommes orientés vers la création d'un module d'authentification permettant d'"enfermer" un utilisateur dans un sous système qui lui est propre où seul le kernel est partagé entre les utilisateurs. Cela nous a permis de réduire de manière drastique la sur-utilisation des ressources dues à la virtualisation mais également réduit les risques de collisions entre deux utilisateurs concurrents.

Malgré l'utilisation de GRSecurity nous permettant de renforcer la base de notre système de virtualisation légère (i.e. chroot), nous n'utilisions pas un système réellement fait pour la sécurité

mais plus pour l'isolation. Nous devions donc nous tourner vers une solution plus pérenne nous permettant une meilleure sécurité via la modification directe des appels systèmes effectués par les machines virtuelles pour éviter tout interférence entre les utilisateurs qu'elles soient volontaires ou involontaires. De plus, notre système de virtualisation légère ne permettait pas de prendre en compte une gestion des ressources par machine virtuelle et nous ne pouvions pas limiter les ressources allouées à chaque utilisateur ce qui risquait de nous amener à des cas où une machine virtuelle vampirise l'ensemble des ressources d'une machine hôte au détriment des autres utilisateurs de la machine. Finalement, nous voulions avoir la capacité de pouvoir suspendre une machine virtuelle et la restaurer afin de limiter l'utilisation des ressources quand un utilisateur est déconnecté. Ce qui nous permettra de déplacer une machine virtuelle d'une machine hôte à l'autre sans pour autant avoir à l'éteindre totalement.

En partant de ces remarques et motivations, nous avons exploré les autres solutions existantes dans le domaine et nous nous sommes tournés vers OpenVZ qui remplissaient l'ensemble de nos demandes.

3.2 Load Balancer

Comme nous l'avions exprimé dans [2], un Load Balancer se basant sur une étude des ressources disponibles sur chaque machine hôte à un moment t, est la meilleure solution dans le cas des bureaux distants car elle permet une répartition de meilleure qualité entre les différentes machines hôtes et prend en compte les changements de comportements des utilisateurs qui impliquent un changement dans leur utilisation des ressources systèmes.

Via les nouveaux paramètres amenés par la virtualisation via OpenVZ, nous avons amélioré notre algorithme afin de prendre en compte des contraintes supplémentaires nous permettant une réparation de charge plus précise. De plus, nous souhaitions pouvoir prendre en compte une fenêtre d'évolution des utilisations ressources de chaque utilisateur. Cette répartition est donc plus précise et de meilleure qualité mais également plus pérenne dans le temps grâce à la prévoyance de ressources suffisante pour un comportement de base de l'utilisateur. De plus, l'ajout des contraintes liées aux machines virtuelles nous permet d'introduire une qualité de service bien supérieure vis-à-vis des ressources disponibles pour chaque utilisateur.

3.3 Live Migration

Dans notre solution précédente ([2]), nous avions discuté de la bonne répartition des ressources à la connexion des utilisateurs mais nous n'avions pas abordé des changements de comportements des utilisateurs impliquant un manque de ressources sur une machine sans qu'aucun nouvel utilisateur ne se soit connecté sur cette machine hôte en même temps. Nous voulions pouvoir gérer cet aspect de changement de comportement d'un ou plusieurs utilisateurs provoquant une sur-utilisation des ressources d'une machine. Pour cela, il nous fallait avoir la capacité de migrer une machine virtuelle d'une machine hôte à une autre. Mais également, nous devions mettre en place un formalisme et un protocole pour gérer cette répartition des machines virtuelles à la volée sur un ensemble de machines. La répartition ne serait plus limitée à la connexion mais à l'ensemble de la connexion de chaque utilisateur. Le but de ce processus de "Live Migration" est de pouvoir s'approcher en permanence du meilleur cas de répartition d'utilisation des ressources.

De plus, la "Live Migration" se doit de n'avoir aucune influence sur l'utilisateur et d'être entièrement transparent pour lui, tout en étant totalement automatisé.

4 Virtualisation

Tout d'abord OpenVZ nous permet de réduire à seulement 1 à 3% la sur- utilisation des ressources par rapport à un système non virtualisé cela le place à la première place dans ce domaine par rapport aux autres solutions de virtualisation. De plus, cette rapidité ne s'accompagne pas d'une perte de fonctionnalités. En effet, chaque machine virtuelle dispose de l'ensemble des capacités d'une machine hôte classique tel qu'un système de fichiers, une interface réseau, etc. Grâce à OpenVZ, nous disposons également de machines virtuelles qui sont totalement isolées les unes des autres (système de fichiers et processus propres, etc), même si elles partagent un kernel commun. Finalement, nous avons remarqué que de nombreuses solutions de virtualisation actuelles peinent à utiliser les fonctionnalités multiprocesseurs des machines actuelles alors que OpenVZ répond ce problème d'une manière efficace grâce à un ordonnanceur permettant de répartir les processus des machines virtuelles sur l'ensemble de la puissance processeur.

L'interface réseau propre à chaque machine virtuelle nous permet de mettre en place des pares-feux pour chaque machine virtuelle mais aussi d'autoriser les interactions entre une machine virtuelle et le reste du monde d'une manière beaucoup plus fine et en adéquation avec les applications installées dans la machine virtuelle. Nous ne risquons plus des collisions sur les interfaces réseaux dus à l'utilisation simultanée de celles-ci par plusieurs utilisateurs dans un même laps de temps grâce à des interfaces réseaux virtuelles propre à chaque VM (Machine Virtuelle). Finalement, cela nous permet également, de créer des réseaux virtuels entre les utilisateurs d'un même groupe ou d'une même entreprise, permettant de délivrer les même services qu'un intranet classique.

Nous voulons également de la qualité de service (QoS) au niveau de l'allocation des ressources afin d'éviter la sur-utilisation du système par un utilisateur en limitant ces ressources. Nous voulons également pouvoir limiter les interférences dans l'usagge des ressources dues à une attaque sur une VM. Grâce à OpenVZ, nous pouvons spécifier le nombre de ressources associées à chaque VM ce qui nous permet en plus d'améliorer par la suite nos algorithmes associées à la répartitions de charge et de surveillance de l'utilisation des ressources.

Afin de pouvoir intégrer OpenVZ dans notre architecture, nous avons développé un nouveau module pour FreeNX, notre serveur de répartition de charge et serveur graphique distant. Ce module prend la forme d'un module de Load Balancing qui va en pratique :

1. Trouver le nom de l'utilisateur.
2. En déduire (via le service d'authentification) l'UID correspondant et l'identifiant de la VM propre à cet utilisateur.
3. Créer, démarrer, réveiller la VM correspondante.
4. Transférer la connexion de l'utilisateur dans la VM.

moment l'ensemble des machines virtuelles est accessible sur chaque machine, c'est pour cela que nous utilisons un NAS pour nos expérimentations et que dans le cas d'une utilisation en production, nous recommandons l'utilisation d'un système de fichiers distribué haute performance.

Afin de permettre un tel processus de répartition de charge, il faut que chaque VM soit disponible à tout moment et dans le même "état" sur chaque noeud de la grille de répartition des machines virtuelles. Pour cela, nous avons commencé par utiliser NFS mais l'ensemble des interactions de systèmes de fichiers ne sont pas disponibles. Cela empêche tout simplement le lancement d'une VM OpenVZ sur une partition NFS. Nous nous sommes donc orientés vers une autre solution permettant de répartir de manière efficace un système de fichiers sur un ensemble de machines tout en permettant un accès à haute vitesse et une disponibilité maximale. Pour cela et ne disposant pas du

matériel hardware nécessaire, nous avons souhaité simuler un SAN. Bien que le fonctionnement soit optimal i.e. que tout fonctionne à la vitesse nécessaire pour notre architecture, nous nous sommes vite rendu compte de l'impossibilité (ou par une augmentation du coût non acceptable) d'un tel dispositif sur une grande implémentation de notre architecture. Finalement, nous nous sommes donc orientés vers un système de fichiers réparti permettant de soutenir une telle disponibilité et vitesse suffisante pour un grand nombre de VM ouvertes en simultané. C'est dans ce cadre que nous avons introduit Lustre qui nous permet de répartir l'ensemble des machines virtuelles des utilisateurs sur l'ensemble de nos noeuds sans nécessité un surcoût prohibitif et en permettant une rapidité et une disponibilité de grande qualité.

Nous disposons donc d'une virtualisation légère nous permettant de limiter au maximum la sur-utilisation de ressources grâce à la virtualisation légère mais également une sécurité accrue, des fonctionnalités supplémentaires telles que la possibilité d'intranet virtuel et de qualité de service.

Comme représenté dans la figure 1, quand un utilisateur se connecte, il est tout d'abord redirigé vers une machine hôte (nodeX) par le Load Balancer (master_node) via un algorithme expliqué dans la section 5.1. Puis nodeX va rechercher dans la liste des machines virtuelles suspendues si celle de l'utilisateur y est présente. Si c'est le cas, nodeX va la restaurer puis mettre à jour la liste des machines virtuelles suspendues et en activité sur la nodeX et informer les autres noeuds de cette modification. Finalement, l'utilisateur est redirigé vers sa machine virtuelle.

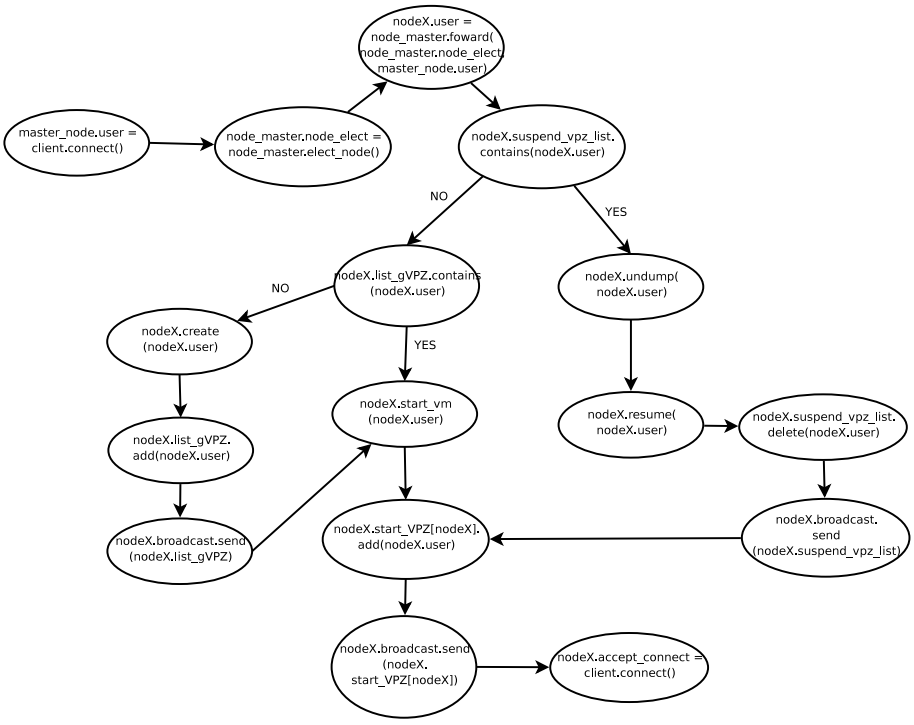

FIG. 1: Graphe représentant la connexion d'un utilisateur

Algorithm 1 Algorithme représentant la déconnexion d'un utilisateur

Require: $userX, nodeX$
1: $nodeX.user = userX$
2: $nodeX.deconnect(nodeX.user)$
3: $nodeX.suspend(nodeX.user)$
4: $nodeX.dump(nodeX.user)$
5: $nodeX.kill(nodeX.user)$
6: $nodeX.suspend\_VPZ\_list.add(nodeX.user)$
7: $nodeX.broadcast.send(nodeX.suspend\_VPZ\_list)$
8: $nodeX.start\_VPZ(nodeX).delete(nodeX.user)$
9: $nodeX.broadcast.send(nodeX.start\_VPZ(nodeX)$

L'algorithme 1 représente ensuite la procédure effectuée lors de la déconnexion d'un utilisateur. Tout d'abord, la machine virtuelle est suspendue et son état global est sauvegardé. Puis la machine arrêtée est ajoutée à la liste des machines virtuelles suspendues, cette mise à jour est envoyée à l'ensemble des autres machines hôtes. Finalement, la VM est enlevée de la liste des machines virtuelles actuellement en marche sur la machine hôte et cette mise à jour est envoyée à l'ensemble des autres machines hôtes. Comme nous pouvons le voir dans la figure 1, chaque machine hôte dispose d'un ensemble de fonctions et variables afin de permettre, faciliter et organiser l'administration des machines virtuelles et leur répartition qui sont listées dans le tableau 1 (voir à la fin du document).

Grâce à ce système, nous limitons l'utilisation de ressources en suspendant une machine dés qu'elle n'est pas utilisée et nous facilitons l'administration via un processus de gestion automatique des machines virtuelles et de réplication de la gestion des machines sur un ensemble de machine augmentant la disponibilité, la tolérance aux pannes ainsi que les performances.

5 Répartition de charge

Un point critique de notre architecture est la répartition des ressources de plusieurs machines hôte à plusieurs machines virtuelles en fournissant toujours le meilleur cas de figure pour l'utilisation des ressources où chaque utilisateur bénéficie de l'ensemble des ressources dont il a besoin pour le fonctionnement normal de son bureau graphique.

En plus de l'architecture proposée dans [2], nous avons mis en place de nouveaux algorithmes de Load Balancing prenant en compte les spécificités des machines virtuelles et nous permettant une répartition de meilleure qualité supportant la qualité de service.

De plus, nous voulons pouvoir à tout moment être dans le meilleur des cas au niveau de l'utilisation des ressources et nous nous retrouvons donc devant le cas où malgré un Load Balancing de qualité, le comportement des utilisateurs peut changer. Nous voulons, pour améliorer l'architecture, pouvoir déplacer des machines virtuelles d'une machine hôte à une autre. Bien entendu, ce "déplacement" doit être totalement transparent pour un utilisateur. Cette "Live Migration" ou Migration à Chaud nous permet d'atteindre l'optimum au niveau de la répartition de charge et donc fournit la plus grande quantité de ressources pour chaque utilisateur à tout moment.

Le Load Balancing comme la "Live Migration" nécessite des algorithmes et des fonctionnements bien précis pour qu'ils soient totalement invisibles pour l'utilisateur tout en amenant les meilleures performances.

5.1 Load Balancing

Comme proposé dans [2], nous avons créé un algorithme se basant sur l'utilisation des ressources à un moment t auquel, nous avons ajouté les paramètres relatifs à chaque VM tournant sur la machine hôte ce qui nous permet une meilleure finesse de notre algorithme et une qualité de service pour les utilisateurs. En effet, nous pouvons vérifier que la machine hôte dispose effectivement de suffisamment de ressources pour fournir à l'utilisateur un bureau graphique totalement fonctionnel pour l'utilisation à laquelle il a besoin i.e. que les ressources allouées minimum à sa VM sont effectivement disponibles sur la machine hôte.

Tout d'abord, une première partie de l'algorithme se repose sur des valeurs statiques propres à chaque machine comme la quantité de RAM. Cette valeur ne va pas changer entre chaque exécution de l'algorithme et est donc stockable pour améliorer la rapidité du Load Balancer. Comme le représente l'équation 1, les quantités de RAM (RAM_{total}), le nombre de processeur (CPU_{number}) et leur rapidité ($CPU_{speed\_in\_Mhz}$ pondérée par une variable fixe α) sont prises en compte. Il est également possible d'ajouter d'autres valeurs statiques (Π) pour prendre en compte d'autres particularités de la machine e.g. la vitesse des disques, la rapidité des bus, etc.

$$\begin{aligned} static\_score = RAM_{total} + \\ CPU_{number} \times (\frac{CPU_{speed\_in\_Mhz}}{\alpha}) + \\ +\Pi \end{aligned} \tag{1}$$

La deuxième partie de l'algorithme se base sur un ensemble de valeurs révélatrices de l'utilisation des ressources d'une machine à un moment t. Cette note dynamique se base sur des valeurs retournées par chaque machine hôte mais également par chaque machine virtuelle. Cette combinaison nous permet d'avoir une note se rapprochant au plus près de la réalité et prenant en compte les spécificités des machines virtuelles.

Elle permet d'améliorer la qualité de service en prenant en compte une double limite (une souple et une rigide) i.e. comme les quotas Unix, le système permet de fixer une première limite (souple) où le système est prévenu en cas de dépassement. Dans ce cas, il autorisera dans la limite des ressources le dépassement puis une second limite (rigide), elle non-dépassable, est fixée avec des valeurs plus grandes. L'algorithme 2 est utilisé aussi bien pour calculer l'utilisation des ressources processeurs (dans ce cas ELEM = CPU) et mémoire (ELEM = RAM) mais pourrait être également utilisé pour d'autres valeurs d'éléments comme le nombre de fichiers ouverts, de bande passante réseaux, etc.

L'algorithme va tout d'abord faire une liste de l'ensemble des machines virtuelles tournant sur la machine hôte dont la note est calculée. Puis pour chaque machine virtuelle (VZ) listée, l'algorithme récupère des valeurs propres à chaque VM correspondant à leur utilisation de ressources ($ELEM_{ru}$) à un moment t ainsi que les valeurs de quota ($ELEM_{hr}$ et $ELEM_{sr}$). Ensuite, en combinant ces valeurs et en calculant les ressources disponibles vis-à-vis de ses quotas pour chaque VM, l'algorithme 2 calcule un score qui va permettre de chiffrer l'utilisation de ressources de la VM. Finalement, ce résultat est pondéré par des variables fixes (x, y, z) et combiné avec le score des autres VM se trouvant sur la machine hôte.

Au final, pour obtenir une note globale, nous multiplions les valeurs trouvées via l'algorithme précédent pour les cas où l'élément (i.e. ELEM) correspond au processeur (i.e. la puissance en Mhz) et à la mémoire (i.e. la RAM) au quel nous ajoutons le nombre de machine virtuelle tournant sur la machine hôte. Cette valeur est pondérée par une variable fixe β.

Algorithm 2 Dynamic Score pour les ELEM : $dscore_{elem}$

for all VZ in VZ_list **do**
  $ELEM_{vz} = 0$
  $ELEM_{ru}$ = numproc utilisé à t
  $ELEM_{hr}$ = numproc réservé et dédié
  $ELEM_{sr}$ = numproc réservé si disponible
  if $ELEM_{ru} < ELEM_{hr}$ **then**
    $ELEM_{hf} = ELEM_{hr} - ELEM_{ru}$
    $ELEM_{sf} = ELEM_{hf} + (ELEM_{sr} - ELEM_{hr})$
  else
    if $ELEM_{ru} = ELEM_{sr}$ **then**
      $ELEM_{flood} = 1$
    end if
    $ELEM_{hf} = -(ELEM_{ru} - ELEM_{hr})$
    $ELEM_{sf} = ELEM_{sr} - ELEM_{ru}$
  end if
  $ELEM_{vz} = y \times ELEM_{hf} + z \times ELEM_{sf} - x \times ELEM_{flood}$
  $ELEM = ELEM + ELEM_{vz}$
end for
$ELEM_{total} = ELEM/fracw \times ELEM_{use}$

$$dynamic\_score = dscore_{ram} \times dscore_{cpu} + \beta \times vz_r \tag{2}$$

Finalement, la note complète de la machine est calculée en multipliant la partie statique et la partie dynamique.

$$score = dynamic\_score \times static\_score \tag{3}$$

La procédure d'élection représentée par la figure 2 montre comme une nodeX va élire une machine hôte (node_elect). Comme on peut le voir, la machine envoie une demande à l'ensemble des autres machines pour qu'elles calculent et lui envoient leur note puis la machine ayant initialisée la procédure d'élection, choisit celle qui a la note la plus élevée. C'est la même procédure que celle qui est utilisée par le Load Balancer pour élire la machine virtuelle ayant le plus de ressources disponibles à un moment t.

5.2 Live Migration

Afin de toujours se rapprocher du meilleur cas au niveau de la répartition des ressources, nous devons pouvoir gérer des modifications de comportement. Par exemple, des utilisateurs provoquant une sur ou sous utilisation d'une machine hôte, ce cas de figure est équilibrable en déplaçant les machines virtuelles en marche à chaud sans perturber le déroulement de la session utilisateur qui s'y trouve. Pour cela, OpenVZ propose des solutions de suspension et de restauration à chaud, nous devons donc les adapter pour mettre en place un système entièrement automatique de répartition à la volée des machines virtuelles et s'adaptant constamment suivant l'utilisation des ressources sur l'ensemble des machines hôte à un moment t. Pour cela, nous avons couplé l'équation de Load Balancer avec des algorithmes de recherche et répartition pour tendre vers le meilleur cas. Nous nous devons aussi de gérer les cas d'ajout et de suppression de machine hôte sur l'architecture et le crash d'une machine hôte.

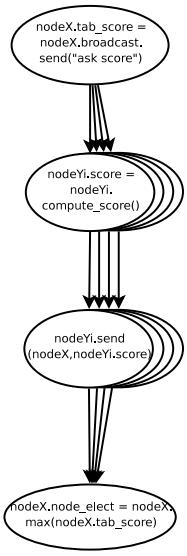

FIG. 2: Graphe représentant la procédure d'éléction

Migration d'une machine virtuelle Tout d'abord, il nous faut pouvoir migrer une machine virtuelle depuis une machine hôte vers une autre. Cette fonctionnalité est représentée par la figure 3. Le but de celle-ci est de migrer une machine virtuelle "nodeX.user" depuis une machine hôte "nodeX" vers une autre "nodeY". Pour commencer, la machine virtuelle est suspendue et son état sauvegardé puis nodeY restaure nodeX.user et prévient nodeX que tout c'est bien passé. Puis nodeX éteint nodeX.user et prévient nodeY que la machine virtuelle est arrêtée, en conséquence, nodeY démarre nodeX.user et prévient nodeX. Ensuite, chacune des machines hôtes met à jour ses listes de machines virtuelles démarrées et prévient les autres machines hôtes.

L'un des avantages de cette fonctionnalité est d'avoir une partie critique, où la machine virtuelle est arrêtée, qui est courte et de bénéficier d'une vérification de l'état de la machine pour confirmer le bon déroulement de la procédure afin qu'une VM ne soit pas arrêtée sur une machine hôte sans avoir été redémarrée sur une seconde.

Demande de migration d'une machine virtuelle Tout d'abord, afin de déterminer si la migration d'une machine virtuelle est possible vers une nouvelle machine hôte, nous avons mis en place un algorithme 3 permettant de calculer l'utilisation de ressources provenant d'une machine virtuelle.

Le but de la procédure de demande de migration est de fournir un ensemble de machines virtuelles list_VPZ en marche sur une machine hôte nodeX et de demander à une deuxième machine hôte nodeY si elle peut accepter une de ces machines virtuelles sans pour autant se surcharger. Comme représenté dans la figure 4, la machine nodeX va parcourir la liste list_VPZ et va proposer une à une les machines virtuelles qu'elle contient. nodeY va calculer sa note actuelle en y ajoutant les ressources utilisées par la machine virtuelle en demande de transfert, si cette note est inférieure à la note représentant la saturation sur une machine alors elle accepte la migration de cette machine

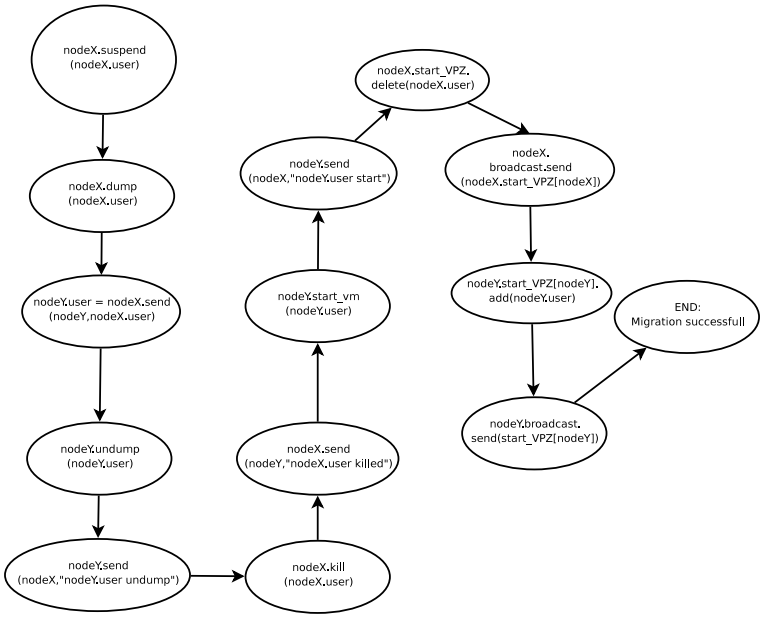

FIG. 3: Graphe représentant la migration d'une machine virtuelle

virtuelle, sinon elle la refuse et nodeX lui propose la suivante, et ainsi de suite jusqu'à ce qu'une machine virtuelle soit sélectionnée ou que la liste soit totalement parcourue.

Répartition à chaud de machine virtuelle C'est une des nouvelles fonctionnalités principales de notre architecture qui permet de répartir de manière optimum et entièrement automatique un ensemble de machines virtuelles sur un ensemble de machines hôtes. La figure 4 montre la procédure qui est lancée toutes les X millisecondes sur chaque machine hôte pour vérifier si elle n'est pas en sur- utilisation et si c'est le cas, demander la migration d'une des machines virtuelles tournant dessus vers une autre machine hôte. Si la machine hôte nodeX est en sur-utilisation, alors elle créé une liste décroissante des machines virtuelles suivant le nombre de ressources qu'elles utilisent et en suivant la condition que si au moins l'une d'entre elles est déplacée alors nodeX ne sera plus en sur-utilisation. Ensuite nodeX va élire un noeud (en utilisant l'algorithme de calcul de note voir algorithme 3) et va essayer d'envoyer la tête de la liste de machines virtuelle vers la node élue jusqu'à ce qu'une VM soit acceptée pour le transfert puis transférée et que finalement, nodeX se retrouve dans un état normal. Si ce n'est pas le cas, nodeX va créer une liste des machines virtuelles. Cette liste est décroissante suivant l'utilisation de ressources des VM et contient celles qui ne suivent pas une condition. Cette condition correspond au postulat : "si l'une des machines virtuelles est déplacées alors la machine hôte ne sera pas en sur-utilisation". Puis comme précédemment, nodeX va essayer de transférer vers la machine hôte ayant le plus de ressources disponibles, chaque VM de la liste. Mais contrairement à la première partie de la procédure, si une machine virtuelle est transférée, nodeX va continuer à demander des transferts de machines virtuelles jusqu'à ce qu'elle atteigne un niveau d'utilisation inférieur à la sur- utilisation. Si, à la fin de la procédure, nodeX est

Algorithm 3 Note d'une machine virtuelle

$CPU_{vz} = 0$
CPU_{ru} = numproc utilisé à t
CPU_{hr} = numproc réservé et dédié
CPU_{hr} = numproc réservé si disponible
if $CPU_{ru} < CPU_{hr}$ **then**
  $CPU_{hf} = CPU_{hr} - CPU_{ru}$
  $CPU_{sf} = CPU_{hf} + (CPU_{sr} - CPU_{hr})$
else
  if $CPU_{ru} = CPU_{sr}$ **then**
    $CPU_{flood} = 1$
  end if
  $CPU_{hf} = -(CPU_{ru} - CPU_{hr})$
  $CPU_{sf} = CPU_{sr} - CPU_{ru}$
end if
$CPU_{vz} = y \times CPU_{hf} + z \times CPU_{sf} - x \times CPU_{flood}$
$RAM_{vz} = 0$
RAM_{ru} = taille de la RAM utilisé à t
RAM_{hr} = taille de la RAM réservé et dédié
RAM_{hr} = taille de la RAM réservé si disponible
if $RAM_{ru} < RAM_{hr}$ **then**
  $RAM_{hf} = RAM_{hr} - RAM_{ru}$
  $RAM_{sf} = RAM_{hf} + (RAM_{sr} - RAM_{hr})$
else
  if $RAM_{ru} = RAM_{sr}$ **then**
    $RAM_{flood} = 1$
  end if
  $RAM_{hf} = -(RAM_{ru} - RAM_{hr})$
  $RAM_{sf} = RAM_{sr} - RAM_{ru}$
end if
$RAM_{vz} = y \times RAM_{hf} + z \times RAM_{sf} - x \times RAM_{flood}$
$score = \alpha \times CPU_{vz} + \beta \times RAM_{vz}$

toujours en sur-utilisation alors nous sommes dans un cas non décidable où la puissance de chaque machine est déjà à son maximum et où la seule solution est d'ajouter de la puissance machine ou de supprimer des utilisateurs.

Ajout d'une machine hôte Afin de pouvoir ajouter de manière dynamique de nouvelles machines hôtes (ou tout simplement supporter le retour d'une machine hôte suite à un redémarrage), nous avons mis en place une procédure automatique présentée par l'algorithme 5. La nouvelle machine hôte nodeX va s'ajouter à la liste des machines hôtes et va prévenir les autres machines hôtes de la modification de cette liste.

Suppression d'une machine hôte Afin de pouvoir gérer de manière dynamique la suppression d'une machine hôte mais également le redémarrage ou l'arrêt de l'une d'entre elles, nous introduisons une fonctionnalité représentée par la figure 5. Tout d'abord, quand la machine hôte nodeX reçoit le signal d'une demande d'arrêt, elle va récupérer la liste de l'ensemble des machines virtuelles tournant actuellement sur elle. Puis pour chacune d'elles, nodeX va élire la machine hôte disposant du plus de ressources et va provoquer la migration de la VM vers cette machine hôte et ainsi de suite jusqu'à qu'il ne reste plus aucune VM en route sur nodeX pour finir sa procédure d'arrêt. Comme il est visible dans cette procédure, nous ne pouvons pas supporter le meilleur des cas et l'arrêt d'une machine peut provoquer une sur- utilisation sur une ou plusieurs machines hôtes mais elle permet qu'aucune VM ne sera stoppée par la procédure d'arrêt d'une machine hôte.

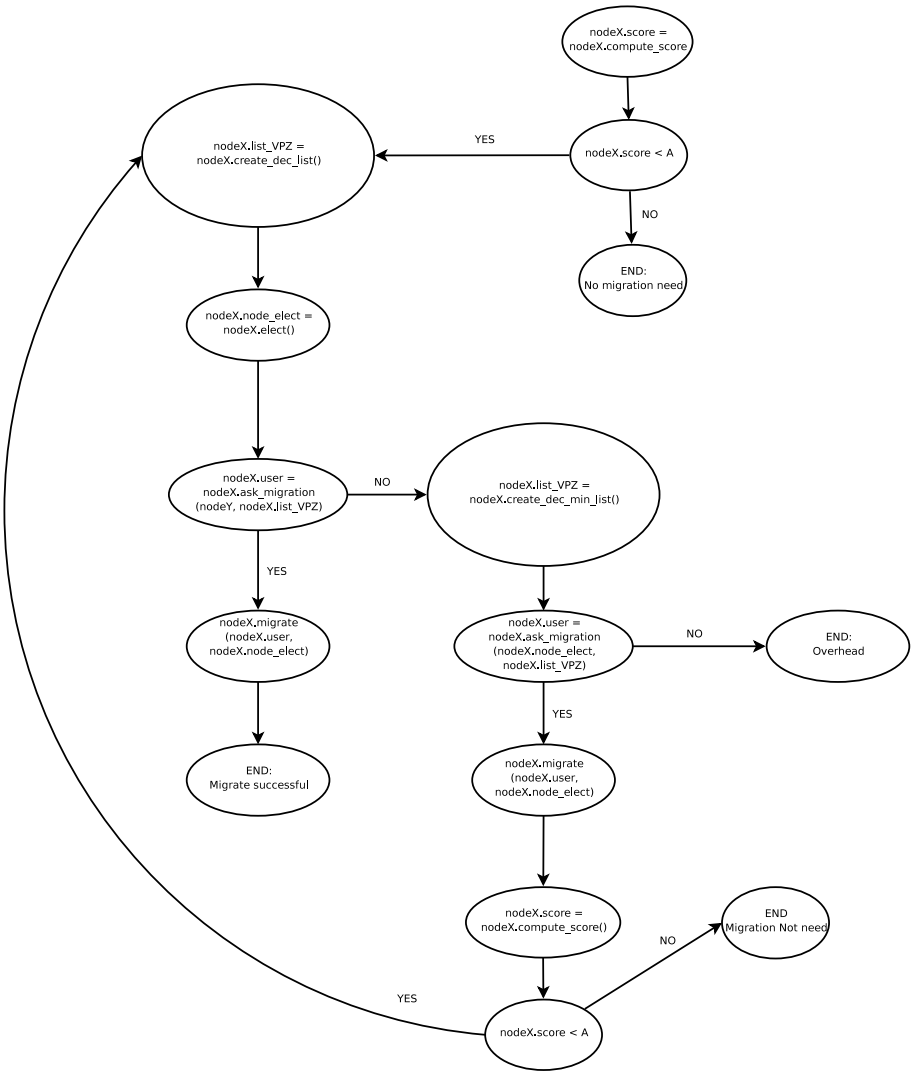

FIG. 4: Graphe représentant la répartition à chaud de machine virtuelle

Algorithm 4 Algorithme représentant demande de migration d'une machine virtuelle

Require: $nodeX, nodeY, nodeX.user$
1: $nodeX.suspend(nodeX.user)$
2: $nodeX.dump(nodeX.user)$
3: $nodeY.user = nodeX.send(nodeY, nodeX.user)$
4: $nodeY.undump(nodeY.user)$
5: $nodeY.send(nodeX, \text{“}nodeY.userundump''\text{)}$
6: $nodeX.kill(nodeX.user)$
7: $nodeX.send(nodeY, \text{“}nodeX.userkilled''\text{)}$
8: $nodeY.start\_vm(nodeY.user)$
9: $nodeY.send(nodeX, \text{“}nodeY.userstart''\text{)}$
10: $nodeX.start\_VPZ.delete(nodeX.user)$
11: $nodeX.broadcast.send(nodeX.start\_VPZ(nodeX))$
12: $nodeY.start\_VPZ.add(nodeY.user)$
13: $nodeY.broadcast.send(nodeY.start\_VPZ(nodeY))$

Algorithm 5 Algorithme représentant l'ajout d'une machine hôte

Require: $nodeX$
1: $nodeX.start()$
2: $nodeX.list_node = nodeX.broadcast.send(\text{“}asklist\_node''\text{)}$
3: $nodeX.list_node = node\_master.send(nodeX, node_master.liste_node)$
4: $nodeX.list_node.add(nodeX)$
5: $nodeX.broadcast.send(list\_node)$

Plantage d'une machine hôte Afin de pouvoir garantir la meilleure continuité possible, nous avons créé une procédure permettant de gérer le plantage d'une machine hôte et la restauration automatique des machines virtuelles qui tournaient dessus (c.f. figure 6). Dans ce cas de figure, un noeud maître "node_master" va supprimer la machine hôte incriminée de la liste des machines hôtes puis envoyer cette information à l'ensemble des autres machines hôtes. Ensuite, il va élire pour chaque VM qui tournaient sur la machine hôte plantée, une nouvelle machine hôte et va lui indiquer quelle machine virtuelle lancer. A la fin de la procédure, toutes les machines virtuelles ont été relancées mais il se peut que certaines machines ne soient plus dans le meilleur des cas i.e. en état de sur-utilisation. Ce cas est non décidable sans l'ajout de nouvelles machines hôtes ou la suppression de certains utilisateurs.

Conclusion Nous avons mis en place un ensemble de procédure et de fonctionnalités, nous permettant de gérer d'une manière totalement automatique un ensemble de machine virtuelles sur un ensemble de machines hôtes en tentant d'atteindre dans tous les cas la meilleure répartition des

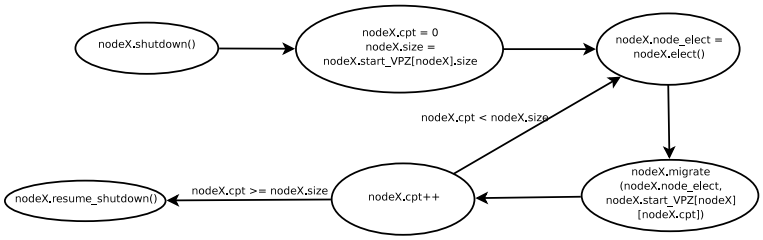

FIG. 5: Graphe représentant la suppression d'une machine hôte

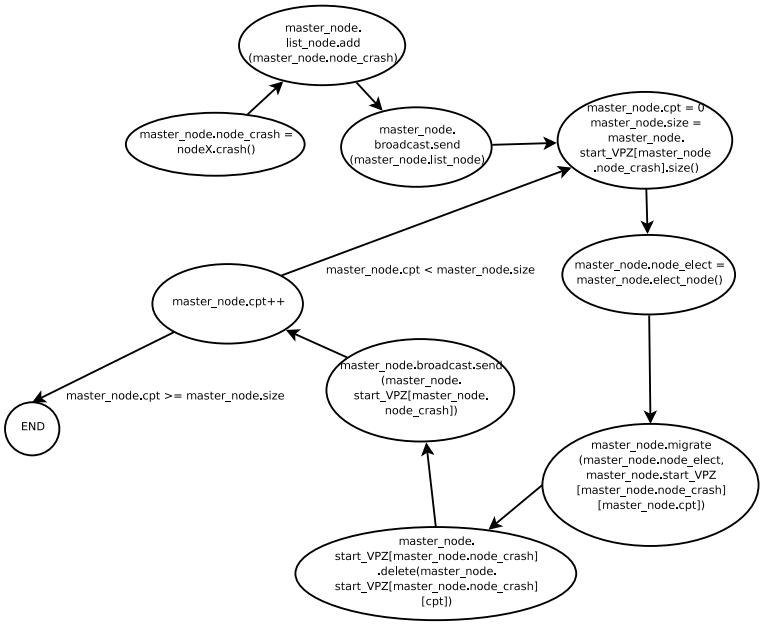

FIG. 6: Graphe représentant le plantage d'une machine hôte

ressources pour qu'elle soit le plus favorable à l'ensemble des utilisateurs tout en supportant une qualité de service.

6 Sécurité

Notre but final étant d'avoir plusieurs utilisateurs (et/ou entreprises) possiblement concurrents sur les mêmes machines, nous devions en plus du partage équitable des ressources et de la séparation des utilisateurs disposer pour notre architecture d'un ensemble d'outils de surveillance au niveau réseau, service ou utilisateur. Pour cela, nous avons utilisé un système à plusieurs couches supportant un ensemble vaste d'outils de sécurité que nous avons étendu pour nos besoins. Certaines parties, comme la corrélation, ont entièrement été conçues et implémentées par nous-même. Ensuite, devant la masse d'information que nous récupérions, nous avions un besoin de centraliser ces informations afin de pouvoir les analyser de manière plus efficace sans pour autant perturber le bon fonctionnement des machines hôtes. C'est cette masse de données, impossible à analyser par l'homme, qui nous a conduit vers des procédures d'automatisations de la corrélation des alarmes entre elles.

6.1 Sécurisation du réseau

Afin de garantir un maximum de sécurité, les points d'entrée de l'architecture i.e. les Load Balancers et les serveurs proxy se trouvent dans une DMZ. Seuls ces serveurs ont un accès à Internet et disposent tous d'un pare-feu personnel permettant uniquement l'accès au service qu'ils proposent.

Les serveurs de services ainsi que l'ensemble des serveurs ne supportant pas un accès direct aux utilisateurs se trouvent dans un réseau à part qui est accessible depuis la DMZ ainsi que depuis le réseau supportant les serveurs où se trouvent les sessions utilisateurs.

Finalement, les serveurs avec les machines virtuelles utilisateurs se trouvent sur un réseau propre. Ce réseau est subdivisé en sous-réseaux virtuels représentant les diverses entreprises se trouvant sur l'architecture (voir section 7).

6.2 Surveillance réseau

Nous avons commencé par mettre en place une observation classique du réseau en utilisant des sondes de détection d'intrusion Snort qui sont reliées à la base centralisée récupérant l'ensemble des rapports de surveillances.

Une première sonde se trouve en tête du réseau et permet l'analyse des attaques envoyées sur l'architecture avant qu'un filtrage n'ait lieu. Nous aurons donc une vue d'ensemble des attaques massives de types scan et/ou DOS qui seront envoyées contre notre réseau ainsi que l'ensemble des packets malicieux qui pourraient être bloqués par nos pare-feu mais qui expliquerait un ralentissement de notre connexion.

Une seconde sonde permet la surveillance de l'architecture comportant les serveurs de Load Balancing et de services ainsi que l'ensemble des serveurs où l'espace utilisateur n'est pas présent mais étant accessible à l'utilisateur. Cette zone spécifique nous permet de définir une politique stricte où seuls les flux reconnus ne lèveront pas d'alarmes.

Finalement, une troisième sonde surveille l'ensemble des noeuds sur lesquels se trouvent les machines virtuelles des utilisateurs et permet de détecter des *botnets* et/ou des utilisations non conformes des sessions utilisateurs qui provoqueraient des traces réseaux.

6.3 Surveillance des services

Afin de pouvoir surveiller les services, nous avons développé des analyseurs qui font lire les fichiers de log générés par les services surveillés et qui vont en générer des rapports. Par exemple, nous disposons d'analyser pour Kerberos, nous permettant de surveiller l'ensemble des demandes, créations, destructions de ticket Kerberos. Cela nous permettra par la suite d'utiliser ces rapports pour la corrélation.

Ce système se base sur des sondes OSSIM qui vont parser les fichiers de logs et envoyer des alertes quand elles trouvent des correspondances avec leur fonction de parsing. Par exemple, la demande de TGT d'un utilisateur va créer un événement qui nous sera utile dans le processus de corrélation. Mais, cela nous permettra également de connaître les mots de passe incorrect ainsi que les potentielles tentatives de brute force sur certains comptes et cela, non plus d'un aspect réseau mais d'un aspect système.

En utilisant le même principe, nous pouvons surveiller l'ensemble du processus de Load Balancing et de Live Migration des machines virtuelles afin de pouvoir s'assurer qu'aucun corruption ou arrêt intempestif ait eu lieu et le cas échéant pouvoir le détecter et y remédier au plus vite.

6.4 Surveillance des systèmes

Tout d'abord, nous disposons d'outils de détection d'intrusion hôte classique tel que OSIRIS afin de surveiller l'intégrité des fichiers qui sont eux même relié à la base centralisée.

L'ensemble des alarmes générées par les diverses sondes i.e. Snort, OSSIM, HIDS sont envoyées à un serveur OSSIM qui va les analyser et vérifier leur validité puis les insérer dans une base de donnée. La connexion entre les sondes et le serveur centralisé OSSIM est sécurisée et utilise des certificats afin de limiter les risques de corruption d'alarmes ou d'insertion d'alarmes factices.

Le volume d'événements et d'alarmes étant assez important et augmentant linéairement avec le nombre d'utilisateurs, une base de donnée tel que MySQL ne supporte pas un tel volume. Nous avons donc opté pour une base de donnée haute performance : IBM DB2. Cela nous permet également de pouvoir prévoir un passage à l'échelle simple et performant grâce à la possibilité de DB2 de se répartir sur un cluster.

La visualisation des alarmes et des événements ainsi que le monitoring des ressources réseaux (via SNMP) et la génération de statistique sur l'état de l'ensemble du réseau est prise en charge par l'interface web d'OSSIM. Cela permet à l'administrateur d'avoir une console centrale pour surveiller l'ensemble du réseau aussi bien du point de vue de sécurité que des ressources.

6.5 Corrélation

Introduction Nous disposons de deux modules de corrélation bien distincts : une validation obligatoire de certains comportements basiques (comme la connexion d'un utilisateur à l'architecture) et une recherche de comportements anormaux/illégaux en analysant l'enchaînement des appels systèmes provoqués par le comportement d'un utilisateur.

Corrélation pour validation d'un comportement normal Comme nous l'avions présenté dans [2], nous utilisons des graphes nous permettant de lier des alarmes et/ou événements suivant un certain nombre de conditions sur leurs valeurs mais également leur chronologie de connexion afin de valider un comportement. Par exemple, nous vérifierons si la connexion d'un utilisateur est bien passée par l'ensemble des services prévus et si tout s'est déroulé correctement ce qui nous permettra de détecter, par exemple, le cas où un utilisateur se connecte directement à une machine hôte sans passer par le Load Balancer ou s'il tente de rejouer un ticket d'authentification.

Nous pouvons également utiliser ce processus de validation pour le Load Balancing et la Live Migration des machines virtuelles afin de pouvoir valider à tout moment les automates introduits dans cet article.

Comme il est visible sur la figure 7, le but de cette signature est de valider le processus de connexion d'un utilisateur. Tout d'abord, une alarme A générée par la connexion d'un utilisateur à l'infrastructure et provenant d'un des Load Balancers. Ensuite une deuxième alarme B signifie l'élection d'une node. Celle-ci a été créée dans un intervalle de 15 secondes après celle de connexion de l'utilisateur et partage avec cette dernière le même hostname et nom d'utilisateur. Ensuite une troisième alarme C est générée signifiant le "fowarding" de la session de l'utilisateur vers le noeud élue. Elle partage avec la précédente le même *hostname*, *nom de noeud élu* et *nom d'utilisateur* et se déroule dans un intervalle de 15 secondes après elle. Une quatrième alarme D représentant la recherche de la VM de l'utilisateur dans la liste des VMs suspendue est crée et elle partage avec C le nom de l'utilisateur, l'hostname de D est égale au nom du noeud élu de C et D a été créé dans un intervalle de 15 secondes après C. Et ainsi de suite afin de valider l'ensemble des comportements de Load Balancing et Live Migration que comporte notre infrastructure.

Corrélation pour la détection d'un comportement anormal Nous disposons d'un ensemble de composants de corrélation nous permettant de reconstruire une session utilisateur qu'elle soit

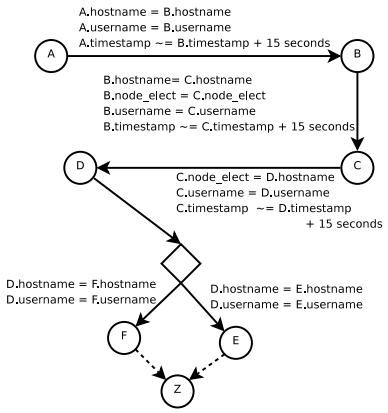

FIG. 7: Exemple d'automate de corrélation pour un comportement normal

locale à une machine, réparties sur un ensemble de manière distribuée i.e. se déroulant en même temps sur plusieurs machines ou de manière séquentielle i.e. se déroulant successivement sur plusieurs machines. Nous pouvons également détecter grâce à ces composants des attaques massives en utilisant des alarmes générées par les sondes systèmes ce qui permet de ne pas se limiter à des attaques massives par réseau mais aussi celle pouvant être lancer localement e.g. attaque par socket Unix. Finalement, un composant nous permet de relier des sessions utilisateurs aux attaques massives que ce soit que la session ait la source de l'attaque ou que la session est été créé suite à une attaque massive.

Ensuite, en nous basant sur un ensemble de signatures représentées par automate, nous analysons ces sessions et nous les classifions. Ces signatures sont générées à partir d'une politique de sécurité à respecter écrite par l'administrateur mais également, grâce à l'analyse des attaques sur un ensemble d'HoneyPot haute-intéraction. Ces travaux ont été publiés dans [1].

7 Un Intranet Virtuel

Chaque machine virtuelle disposant de sa propre interface réseau, nous pouvons proposer des réseaux virtuels internes à un groupe d'utilisateurs avec leurs services propres en découlant i.e. webservice, partage de fichiers communs, etc.

Via l'utilisation de VLAN, nous pouvons proposer à un groupe un réseau intranet complet ainsi que l'ensemble des services qui peuvent s'y trouver et cela, sans risque d'interférence avec les autres groupes se trouvant sur le réseau.

8 Conclusion

Depuis la publication de [2], notre architecture a grandement évolué nous permettant des améliorations importantes dans le domaine de la virtualisation avec l'acceptation de OpenVZ qui nous a permis une meilleure sécurité avec un isolement de meilleure qualité des utilisateurs, mais également une gestion plus poussée des ressources ainsi que des possibilités nouvelles. Ce sont grâce

à ces nouvelles fonctionnalités que nous avons pu améliorer notre structure de Load Balancing en prenant en compte des variables supplémentaires, ce qui nous permet une meilleure réparation des sessions graphiques dès la connexion. Mais également la prise en compte d'une double limite sur l'utilisation des ressources par chaque utilisateur nous permettant d'inclure un aspect de qualité de service garanti.

OpenVZ nous a également permis de mettre en place une structure complète et entièrement automatique de "Live-Migration". Ce qui nous a permit de gérer efficacement les changements de comportements des utilisateurs, tout en augmentant la disponibilité de la plate-forme complète. Les machines virtuelles ne dépendent plus des machines hôtes et nous pouvons de manière totalement dynamique et automatisée augmenter ou réduire la puissance disponible pour la totalité des machines virtuelles.

En combinant de nombreux types de senseurs de surveillance et monitoring mais également deux méthodes de corrélation, nous disposons d'une surveillance et d'une sécurité accrues pour notre architecture. Cela permet d'avoir une vision en temps réel du bon déroulement de l'ensemble des processus sur l'architecture et de connaître et détecter rapidement les points faibles du système global.

Finalement, bien qu'un travail de développement et d'implémentation conséquent reste à faire, nous disposons d'une architecture conceptuelle nous permettant de créer un SI complet comprenant intranet, services et bureaux graphiques sur une architecture distante ayant un haut niveau de sécurité et de disponibilité. De plus, plusieurs SI peuvent cohabiter sur la même architecture sans risque de fuite d'informations entre eux.

Toutes les outils, applications et améliorations qui sont développés pour ce projet, sont intégralement disponibles, au fur et à mesure de leur avancement sous licence GPL.

9 Avancées Futures

Il reste encore de nombreux composants à implémenter mais la structure globale fonctionne déjà bien que ne disposant pas de toutes les fonctionnalités.

L'une des avancées qui nous tient à coeur, est la mise en place d'un système de contrôle d'accès de type MAC mais pour le moment, aucun n'est disponible pouvant fonctionner avec OpenVZ. Nous allons donc étudier comment implémenter une telle fonctionnalité dans notre architecture. Une de nos implémentations future est de pouvoir adapter notre outil à l'analyse de trace système de type SystemTap qui nous permettra de surveiller des propriétés de sécurité telles que les flux de données, la non-interférence, l'intégrité, la confidentialité ou l'abus de privilége. Bien entendu, contrairement aux systèmes MAC, nous ne pourrons pas obliger l'application d'une partie de ces propriétés de sécurité, nous ne pourrons que les surveiller. En effet, les possibilités d'obliger le respect de propriété de sécurité sous les systèmes de contrôle d'accès DAC sont très limités.

Actuellement, le code permettant la "Live Migration" n'est pas encore utilisable dans un environnement de production. Un travail de perfectionnement de ce support et de validation en dehors de notre plate-forme de développement est encore essentielle.

Pour le moment, nous ne disposons que d'un sous-ensemble des signatures de corrélation de comportement valide pour l'ensemble des services de notre architecture. Avec l'augmentation des services disponibles, le nombre de signatures risquent d'augmenter exponentiellement et il serait donc intéressant de réfléchir à des techniques d'apprentissage automatique plus ou moins supervisé pour aider l'administrateur dans l'écriture de ces signatures.

L'ensemble de nos tests et de nos implémentations ont été fait sur une architecture de développement. Nous n'avons donc pas pu essayer dans un milieu réel le passage à l'échelle de l'ensemble de nos logiciels. De plus, nous sommes les seuls à tester cette architecture et un retour sur expérience d'utilisateurs naïfs serait très intéressant.

Des acquisitions de matériels récents vont permettre de nouveaux tests dans des cas d'utilisation réels (i.e. bureautique et TP).

10 Remerciements

Vincent LE-PORT et Kevin GUERIN.

Etiquette	***Description***
noeuds	
$list\_node$	Liste contenant le hostname de l'ensemble des noeuds
machines virtuelles supplémentaire	
$start\_VPZ$	Matrice contenant l'ensemble des machines virtuelles démarrées par noeud
$list\_gVPZ$	Liste contenant l'ensemble des machines virtuelles
$suspend\_VPZ\_list$	Liste contenant l'ensemble des machines virtuelles suspendues
$start()$	Fonction signifiant le démarrage du noeud
$broadcast.send(msg)$	Fonction permettant d'envoyer un message msg à l'ensemble des noeuds
$send(dest\_node, msg)$	Fonction permettant d'envoyer un message msg au noeud $dest\_node$
$compute\_score()$	Fonction de calcul du score actuel du noeud
$compute\_score(node, user)$	Fonction calculant le score actuel d'un noeud $node$ en y ajoute une machine virtuelle appartenant à $user$
$compute\_score(user)$	Fonction retournant le score d'une machine virtuelle $user$
$create\_dec\_list()$	Fonction retournant la liste des machines virtuelles présentes sur le noeud et respectant l'équation suivante $score - compute\_score(user) > A$
$create\_dec\_min\_list()$	Fonction retournant la liste des machines virtuelles présentes sur le noeud et ne respectant pas l'équation suivante $score - compute\_score(user) > A$
$ask\_migration(node, user)$	Demande au noeud $node$ si il accepte une nouvelle machine virtuelle $user$
$shutdown()$	Fonction signifiant la procédure d'arrêt du noeud
$elect\_node()$	Fonction d'élection d'un noeud
$resume\_shutdown()$	Fonction permettant de lancer l'arrêt total de la machine
$suspend(user)$	Procédure permettant de suspendre une machine virtuelle $user$
$dump(user)$	Procédure lançant la sauvegarde des variables dynamiques de la machine virtuelle $user$
$undump(user)$	Procédure permettant de restaurer les variables dynamiques de la machine virtuelle $user$
$resume(user)$	Procédure permettant de relancer une machine virtuelle $user$ suspendu
$kill(user)$	Procédure permettant de stopper une machine virtuelle $user$
$start\_vm(user)$	Procédure permettant de démarrer une machine virtuelle $user$
$create(user)$	Création d'une machine virtuelle pour l'utilisateur $user$
$crash()$	Fonction signifiant le crash du noeud
$foward(node, user)$	Transfère la connexion de l'utilisateur $user$ sur le noeud $node$

TAB. 1: Fonction et variables d'un objet node

Références

1. J. Rouzaud-Cornabas, P. Clemente, and C. Toinard. Correlation of system events : High performance classification of selinux activities and scenarios. In Ratan Kumar Guha and Luca Spalazzi, editors, *Workshop on Security and High Performance Computing Systems*, Cyprus, 2008. ECMS.
2. J. Rouzaud-Cornabas and N. Viot. Secured architecture for remote virtual desktops. In Waleed W. Smari and William McQuay, editors, *Workshop on Collaboration and Security 2007*, Orlando, 2007. IEEE Computer Society.
3. Globus Team. Talks and articles about globus, 2003-2007. http ://workspace.globus.org/papers/index.html.
4. IBM Sun Ray Team. Sun ray, 2007. http ://sun.com/sunray/.
5. OpenVZ Team. Unique features of openvz, 2006. http ://openvz.org/documentation/tech/features.